MQL4 Reference

MetaQuotes Language 4 (MQL4) is a built-in language for programming trading strategies. This language is developed by MetaQuotes Software Corp. based on their long experience in the creation of online trading platforms. Using this language, you can create your own Expert Advisors that make trading management automated and are perfectly suitable for implementing your own trading strategies. Besides, using MQL4 you can create your own technical indicators (custom indicators), scripts and libraries.

MQL4 contains a large number of functions necessary for analyzing current and previously received quotes, and has built-in basic indicators and functions for managing trade orders and controlling them. The MetaEditor (text editor) that highlights different constructions of MQL4 language is used for writing the program code. It helps users to orientate themselves in the expert system text quite easily.

The brief guide contains functions, operations, reserved words, and other language constructions divided into categories, and allows finding the description of every used element of the language.

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

- **Expert Advisor** is a mechanical trading system linked up to a certain chart. An Expert Advisor starts to run when an event happens that can be handled by it: events of initialization and deinitialization, event of a new tick receipt, a timer event, depth of market changing event, chart event and custom events. An Expert Advisor can both inform you about a possibility to trade and automatically trade on an account sending orders directly to a trade server. Expert Advisors are stored in *terminal_directory\MQL4\Experts*.
- **Custom Indicator** is a technical indicator written independently in addition to those already integrated into the client terminal. Like built-in indicators, they cannot trade automatically and are intended for implementing of analytical functions only.
  Custom indicators are stored in *terminal_directory\MQL4\Indicators*
- **Script** is a program intended for a single execution of some actions. Unlike Expert Advisors, scripts do not process any actions, except for the start event (this requires the OnStart handler function in a script). Scripts are stored in *terminal_directory\MQL4\Scripts*
- **Library** is a set of custom functions intended for storing and distributing

frequently used blocks of custom programs. Libraries cannot start executing by themselves.

Libraries are stored in *terminal_directory\MQL4\Libraries*

- **Include File** is a source text of the most frequently used blocks of custom programs. Such files can be included into the source texts of Expert Advisors, scripts, custom indicators, and libraries at the compiling stage. The use of included files is more preferable than the use of libraries because of additional burden occurring at calling library functions.

  Include files can be stored in the same directory as a source file - in this case the #include directive with double quotes is used. Another place to store include files is *terminal_directory\MQL4\Include, in this case the* #include directive is used with angle brackets.

What's New in MQL4

Starting from build 600, MQL4 programming language has been completely revised reaching the level of MQL5 - now you can develop trading robots in MQL4/5 using the unified MetaEditor development environment, single style, libraries and debugging tools.

MQL4 is popular among automated system developers due to the ease of learning and a huge amount of code generated for many years of using MetaTrader 4 terminal. However, the language also has some drawbacks arising from its main advantage - a simple programming language does not allow development of complex systems and hinders porting of debugged libraries from high-level languages. Therefore, we decided to implement in it the maximum possible amount of MQL5 language functions and features fully preserving MQL4 functionality. In other words, all powerful MQL5 functions, including OOP and the native code compiler, will become available in MQL4.

To achieve this, we have developed a unified compiler that automatically supports both MQL4 and MQL5 languages. MetaEditor will also become a unified application both for MetaTrader 4 and MetaTrader 5 platforms. Thus, it will be possible to compile both MQL4 and MQL5 from any version. MQL5 Storage also becomes available for work.

Protection of MQL4 applications rises to MQL5 level. New EX4/EX5 files are provided with a serious and completely revised protection. This means that the Market of secure EX4 applications also becomes available to MetaTrader 4.

Besides, MQL4 now features new graphical objects and new functions for working with charts. MQL5 Standard Library is to be ported to MQL4 providing developers with new possibilities in creating their own graphical interfaces and trading libraries. Now, you can create full-fledged applications in MetaTrader 4 using the resources.

Changes in MQL4 Language

Added new char, short, long, uchar, ushort, uint, ulong and double data types. This will allow transferring codes from other C++ like languages. Data of various types is processed at different rates. Integer data is the fastest one to be processed. A special co-processor is used to handle the double-precision data. However, due to the complexity of the internal representation of floating-point data, it is processed slower than integer one. Typecasting has also been implemented.

Strings are now presented in Unicode format, though they were in ANSI format (single byte ones) before. That should be considered if the program uses DLLs and passes string variables to them.

Predefined Volume variable is now of long type. The time series for accessing the volumes also consist of long type arrays. It is recommended to use explicit casting of data having this type to the target type in old MQL4 programs to avoid type overflow error.

Structures and classes, object pointers, void type and this key word allowing an object to receive a reference to itself have been added. All object-oriented programming standards are supported:

· Encapsulation and Extensibility of Types

· Inheritance

· Polymorphism

· Overload

· Virtual functions

OOP allows developing programs using classes. This facilitates debugging and development of large applications, as well as provides ability to reuse previously generated code multiple times due to inheritance. However, that does not mean that you cannot generate your MQL4 code in procedure-oriented style as before. You can develop your programs as you did in the past if you don't need the new features.

init(), deinit() and start() predefined functions have remained for compatibility, however, OnInit(), OnDeinit(), OnStart(), OnCalculate() and OnTick() ones can now be used instead. Besides, new predefined OnTimer(), OnChartEvent() and OnTester() handler functions have been implemented. In the previous MQL4, predefined functions could have any parameters and any return type, and they could be called by their names, not signatures. In the new MQL4, all predefined functions should strictly correspond to their signatures. In other words, they should have precisely defined set of parameters and return type.

Now, variable names cannot contain special characters and points, and new MQL4 language keywords cannot be used as names. Old MQL4 programs can be recompiled with the new compiler in order to easily correct all such errors while following the compiler's messages.

The Precedence rule now matches C language standards. If you are unsure, you can insert parentheses in old MQL4 apps to clearly indicate the priority to increase reliability.

Shortened conditions check is now used in logical operations, unlike the old MQL4 version where all expressions have been calculated and the check has been performed afterwards. Suppose there is a check of two conditions with the use of logical AND:

```
  if(condition1 && condition2)
    {
     // some block of statements
    }
```

If condition1 expression is false, calculation of condition2 expression is not performed, as false && true result is still equal to false.

ArrayCopyRates() function behavior has changed. In the previous MQL4 versions this function was used for copying price series to array double[][6]. Now, if you need to receive a time series, use the arrays of the MqlRates structure elements:

```
//--- Structure that stores information about the prices, volumes and spre
struct MqlRates
  {
   datetime time;         // Period start time
   double   open;         // Open price
   double   high;         // The highest price of the period
   double   low;          // The lowest price of the period
   double   close;        // Close price
   long     tick_volume;  // Tick volume
   int      spread;       // Spread
   long     real_volume;  // Trade volume
  };
```

Also the new function format can be used for virtual copying, when there is no actual copying, and accessing the copied values you actually access the price data.

```
int  ArrayCopyRates(
   MqlRates&  rates_array[],   // MqlRates array, passed by reference
   string     symbol=NULL,     // symbol
   int        timeframe=0      // timeframe
   );
```

To provide compatibility with old MQL4 programs, the previous call format is also preserved, but now it leads to actual copying of data into a double-type array.

```
int  ArrayCopyRates(
   void&      dest_array[][],     // destination array, passed by reference
   string     symbol=NULL,        // symbol
   int        timeframe=0         // timeframe
   );
```

This means that when the values in the time series change (new bars are added, restructuring, the last bar's Close price is uodated), you must re-copy the required data into the dest_array[][]. The receiver array will be automatically allocated for the required number of copied bars, even if it was declared statically.

Changed RateInfo history data storage format. RateInfo structure was presented as follows in the old version:

```
struct RateInfo
  {
   unsigned int       ctm;   // bar open date
   double             open;  // Open price
   double             low;   // Low price
   double             high;  // High price
   double             close; // Close price
   double             vol;   // volume
  };
```

In the new format, RateInfo structure features fields for storing spread and trading volume:

```
//--- Standard quote presentation in the new terminal version
struct RateInfo
  {
   INT64              ctm;               // open date and time
   double             open;              // Open price (absolute value)
   double             high;              // Low price
   double             low;               // High price
   double             close;             // Close price
   UINT64             vol;               // tick volume
   INT32              spread;            // spread
   UINT64             real;              // trade volume
  };
```

Thus, if MQL4 programs contain DLLs for passing/accepting price data, the corresponding functions in the source codes should be rewritten and recompiled considering format changes to ensure proper operation.

Old EX4 applications and DLLs based on old RateInfo format will not work in the new terminal. Conversion to the new format is required.

In file operations, the number of simultaneously opened files can now reach 64 ones, while there could be no more than 32 ones in the old MQL4. Until recently, the files were always opened in FILE_SHARE_READ or FILE_SHARE_WRITE mode. Now, the necessary opening mode should be specified explicitly.

For FileWrite(), FileWriteArray(), FileWriteDouble(), FileWriteInteger() and FileWriteString() functions the type of returned value has been changed from int to uint. The functions return the number of bytes, actually written or 0 in case of error (in old version of MQL4 the negative number is returned in case of error).

Working with functions, scope of variables and memory release in local arrays has also been changed. Since the number of changes is large enough, the new #property strict property has been introduced to provide maximum compatibility with the previous approach to developing MQL4 programs. When creating new MQL4 application using MQL wizard, this property is always added to the template.

The string representation of datetime type depends on compilation mode:

```
   datetime date=D'2014.03.05 15:46:58';
   string str="mydate="+date;
//--- str="mydate=1394034418" - old compiler/new compiler without #propert
//--- str="mydate=2014.03.05 15:46:58" - new compiler with #property stric
```

The table below contains the differences between MQL4, new MQL4 without using strict and new MQL4 with specified strict compilation mode:

```
#property strict
```

When compiling libraries in the strict mode, the export modifier should be added for each exported function, otherwise the function will not be accessible from outside.

The table of differences between compilers:

Old MQL4 compiler	New MQL4 compiler	New MQL4 with #property strict
init(), start() and deinit()	init(), start() and deinit()	Ditto

entry points may have any parameters and any return type	have been remained intact for compatibility, while new OnInit(), OnStart(), OnCalculate(), OnTick(), OnTimer(), OnChartEvent(), OnTester() and OnDeinit() should strictly correspond to their signatures	
The result of the return from init() function is not analyzed by the runtime subsystem	The result of the return from init() and OnInit() functions is not analyzed by the runtime subsystem	If a non-zero value is returned from OnInit(), the operation of an Expert Advisor or an indicator is stopped, the program is unloaded
Virtually any variable names (except for the reserved words) are possible, including special characters and points	Variable names cannot have special characters and points. The list of the reserved words has been expanded. Thus, such widespread words as short, long, const, etc. cannot be used as names	Ditto
Variable scope is from declaration (even in the nested block) to the function end	Ditto	Variable scope is from declaration to the end of the block, in which the variable is declared
Implicit initialization of all the variables (both global and local ones) by zero	Ditto	Only global variables are initialized. In local variables, only strings are initialized implicitly
Local arrays are not released when exiting the function	Local arrays are released when exiting the function	Local arrays are released when exiting {} block
"Array out of range" does not cause a critical error	Ditto, except for the arrays of structures and classes, for which this error is critical one	**"Array out of range"** is a critical error causing the program to stop
No structures and classes	Structures and classes are present. Additional data types are implemented	Ditto

Strings are single-byte. datetime is a 32-bit integer Predefined Volume variable is of double type	Strings are unicode ones. datetime is a 64-bit integer Predefined Volume variable is of long type	Ditto
ArrayCopyRates() performs virtual copying to double[][6] array	ArrayCopyRates() performs virtual copying to MqlRates[] array. Copying to double[][6] array has remained intact for the sake of compatibility, however, that copying is real, not virtual.	Ditto
The functions may not return values even if they have a type. To do this, return(0) is automatically inserted by the compiler in the function end	Ditto	Functions of any type should return a value
The number of simultaneously opened files is 32	The number of simultaneously opened files is 64	Ditto
The files are always opened in FILE_SHARE_READ, FILE_SHARE_WRITE mode **	FILE_SHARE_READ and/or FILE_SHARE_WRITE should be specified explicitly	Ditto
The names of extern variables are displayed for scripts in the input parameters window	The names of extern and input variables are displayed for scripts in show_inputs mode in the input parameters window	String comments instead of extern and input variable names are displayed for scripts in show_inputs mode in the input parameters window

* Please pay special attention to "Array out of range" error - many old custom indicators will display this error in strict mode of the new compiler when launched on the chart. It is recommended to find the cause and eliminate it.

** In the new MQL4 and MQL5, FILE_SHARE_READ and FILE_SHARE_WRITE flags are responsible for the files shared use mode. There were no such files in the old MQL4.

Changes in File Structure

In the previous builds of MetaTrader 4 client terminal (509 and older), all MQL4 applications were stored in the following subdirectories of **<terminal_installation_folder>\experts\** root directory:

- \experts - Expert Advisors (trading robots),
- \experts\indicators - custom indicators,
- \experts\scripts - scripts (MQL4 applications for a single run on the chart),
- \include - source code MQH and MQ4 files implemented into other programs,
- \libraries - libraries in the form of MQ4 source codes and EX4 executable files compiled from them. They are used for the dynamic call of the functions contained there by other MQL4 programs,
- \files - special "file sandbox". MQL4 applications are allowed to execute file operations only within this directory.

In the new MQL4 version, the file structure for storing the source codes has changed. Now, all MQL4 applications should be located in the appropriate folders of **<data_folder>\MQL4\** directory:

- \Experts - Expert Advisors (trading robots),
- \Indicators - custom indicators,
- \Scripts - scripts (MQL4 applications for a single run on the chart),
- \Include - source code MQH and MQ4 files implemented into other programs,
- \Libraries - libraries in the form of MQ4 source codes and EX4 executable files compiled from them. They are used for the dynamic call of the functions contained there by other MQL4 programs,
- \Images - image files for using in resources,
- \Files - special "file sandbox". MQL4 applications are allowed to execute file operations only within this directory.

When updating MetaTrader 4 terminal from build 509 to the newer version, all MQ4, MQH and EX4 files from standard root directories of the previous version are automatically copied and relocated to the appropriate folders. **Subfolders additionally created by a user, as well as files contained there are not processed.** They should be relocated to the new place manually if necessary.

> **No files or folders are deleted during the update!** All file copy operations including used file paths are fixed in the terminal Journal during the update.

No automatic re-compilation of the old EX4 files to the new version is

performed during the update. Users are free to decide what source codes should be compiled to the new EX4 version. All old EX4 will work in the new MetaTrader 4 terminal. EX4 libraries compiled by the new compiler can be called only from the EX4 programs that have also been compiled in the new version.

In some cases, you may need to edit the path in #include for included files (if relative paths have changed) in the source files. Please note that MetaEditor's root directory is now <data_folder>\MQL4\. All programs should be located in the correct subdirectories.

You can find the data folder (<data_folder>) for each copy of MetaTrader 4 terminal on your computer via the terminal menu or in MetaEditor: File - Open Data Folder.

Language Basics

The MetaQuotes Language 4 (MQL4) is an object-oriented high-level programming language intended for writing automated trading strategies, custom technical indicators for the analysis of various financial markets. It allows not only to write a variety of expert systems, designed to operate in real time, but also create their own graphical tools to help you make trade decisions.

MQL4 is based on the concept of the popular programming language C++. The language has enumerations, structures, classes and event handling. By increasing the number of embedded main types, the interaction of executable programs in MQL4 with other applications through dll is now as easy as possible. MQL4 syntax is similar to the syntax of C++, and this makes it easy to translate into it programs from modern programming languages.

To help you study the MQL4 language, all topics are grouped into the following sections:

Syntax

As to the syntax, THE MQL4 language for programming trading strategies is very much similar to the C++ programming language, except for some features:

· no address arithmetic;

· no goto operator;

· an anonymous enumeration can't be declared;

· no multiple inheritance.

See also

Enumerations, Structures and Classes, Inheritance

Comments

Multi-line comments start with the /* pair of symbols and end with the */ one. Such kind of comments cannot be nested. Single-line comments begin with the // pair of symbols and end with the newline character, they can be nested in other multi-line comments. Comments are allowed everywhere where the spaces are allowed, they can have any number of spaces in them.

Examples:

```
//--- Single-line comment /*  Multi-
    line         // Nested single-line comment
    comment
*/
```

Identifiers

Identifiers are used as names of variables and functions. The length of the identifier can not exceed 63 characters.

Characters allowed to be written in an identifier: figures 0-9, the Latin uppercase and lowercase letters a-z and A-Z, recognized as different characters, the underscore character (_).The first character can not be a digit.

The identifier must not coincide with reserved word.

Examples:

```
NAME1 namel Total_5 Paper
```

See also

Variables, Functions

Reserved Words

The following identifiers are recorded as reserved words, each of them corresponds to a certain action, and cannot be used in another meaning:

Data Types

bool	enum	struct
char	float	uchar
class	int	uint
color	long	ulong
datetime	short	ushort
double	string	void

Access Specificators

const	private	protected
public	virtual	

Memory Classes

extern	input	static

Operators

break	dynamic_cast	return
case	else	sizeof
continue	for	switch
default	if	while
delete	new	
do	operator	

Other

false	#define	#property

this	#import	template
true	#include	typename
strict		

Data Types

Any program operates with data. Data can be of different types depending on their purposes. For example, integer data are used to access to array components. Price data belong to those of double precision with floating point. This is related to the fact that no special data type for price data is provided in MQL4.

Data of different types are processed with different rates. Integer data are processed at the fastest. To process the double precision data, a special co-processor is used. However, because of complexity of internal representation of data with floating point, they are processed slower than the integer ones.

String data are processed at the longest because of dynamic computer memory allocation/reallocation.

The basic data types are:

- integers (char, short, int, long, uchar, ushort, uint, ulong);
- logical (bool);
- literals (ushort);
- strings (string);
- floating-point numbers (double, float);
- color (color);
- date and time (datetime);
- enumerations (enum).

Complex data types are:

- structures;
- classes.

In terms of OOP complex data types are called abstract data types.

The *color* and *datetime* types make sense only to facilitate visualization and input of parameters defined from outside - from the table of Expert Advisor or custom indicator properties (the Inputs tab). Data of *color* and *datetime* types are represented as integers. Integer types and floating-point types are called arithmetic (numeric) types.

Only implicit type casting is used in expressions, unless the explicit casting is specified.

See also

Typecasting

Integer Types

In MQL4 integers are represented by eleven types. Some types can be used together with other ones, if required by the program logic, but in this case it's necessary to remember the rules of typecasting..

The table below lists the characteristics of each type. Besides, the last column features a type in C++ corresponding to each type.

Type	Size in Bytes	Minimum Value	Maximum Value	C++ Analog
char	1	-128	127	char
uchar	1	0	255	unsigned char, BYTE
bool	1	0(false)	1(true)	bool
short	2	-32 768	32 767	short, wchar_t
ushort	2	0	65 535	unsigned short, WORD
int	4	- 2 147 483 648	2 147 483 647	int
uint	4	0	4 294 967 295	unsigned int, DWORD
color	4	-1	16 777 215	int, COLORREF
long	8	-9 223 372 036 854 775 808	9 223 372 036 854 775 807	__int64
ulong	8	0	18 446 744 073 709 551 615	unsigned __int64
datetime	8	0 (1970.01.01 0:00:00)	32 535 244 799 (3000.12.31 23:59:59)	__time64_t

Integer type values can also be presented as numeric constants, color literals, date-time literals, character constants and enumerations.

See also

Conversion Functions, Numeric Type Constants

Char, Short, Int and Long Types

char

The *char* type takes 1 byte of memory (8 bits) and allows expressing in the binary notation 2^8=256 values. The *char* type can contain both positive and negative values. The range of values is from -128 to 127.

uchar

The *uchar* integer type also occupies 1 byte of memory, as well as the *char* type, but unlike it *uchar* is intended only for positive values. The minimum value is zero, the maximum value is 255. The first letter u in the name of the *uchar* type is the abbreviation for *unsigned*.

short

The size of the *short* type is 2 bytes (16 bits) and, accordingly, it allows expressing the range of values equal to 2 to the power 16: 2^16 = 65 536.Since the *short* type is a signed one, and contains both positive and negative values, the range of values is between -32 768 and 32 767.

ushort

The unsigned *short* type is the type *ushort*, which also has a size of 2 bytes. The minimum value is 0, the maximum value is 65 535.

int

The size of the *int* type is 4 bytes (32 bits). The minimal value is -2 147 483 648, the maximal one is 2 147 483 647.

uint

The unsigned integer type is *uint*. It takes 4 bytes of memory and allows expressing integers from 0 to 4 294 967 295.

long

The size of the *long* type is 8 bytes (64 bits). The minimum value is -9 223 372 036 854 775 808, the maximum value is 9 223 372 036 854 775 807.

ulong

The *ulong* type also occupies 8 bytes and can store values from 0 to 18 446 744 073 709 551 615.

Examples:

```
char  ch=12; short sh=-5000;
int   in=2445777;
```

Since the unsigned integer types are not designed for storing negative values, the attempt to set a negative value can lead to unexpected consequences. Such a simple script will lead to an infinite loop:

```
//--- Infinite loop
void OnStart()
  {
   uchar  u_ch;

   for(char ch=-128;ch<128;ch++)
     {
      u_ch=ch;
      Print("ch = ",ch," u_ch = ",u_ch);
     }
  }
```

The correct variant is:

```
//--- Correct variant
void OnStart()
  {
   uchar  u_ch;

   for(char ch=-128;ch<=127;ch++)
     {
      u_ch=ch;
      Print("ch = ",ch," u_ch = ",u_ch);
      if(ch==127) break;
     }
  }
```

Result:

```
   ch= -128  u_ch= 128
   ch= -127  u_ch= 129
   ch= -126  u_ch= 130
   ch= -125  u_ch= 131
   ch= -124  u_ch= 132
   ch= -123  u_ch= 133
   ch= -122  u_ch= 134
   ch= -121  u_ch= 135
   ch= -120  u_ch= 136
   ch= -119  u_ch= 137
   ch= -118  u_ch= 138
   ch= -117  u_ch= 139
   ch= -116  u_ch= 140
   ch= -115  u_ch= 141
   ch= -114  u_ch= 142
   ch= -113  u_ch= 143
   ch= -112  u_ch= 144
   ch= -111  u_ch= 145
    ...
```

Examples:

```
//--- Negative values can not be stored in unsigned types
uchar  u_ch=-120;
ushort u_sh=-5000;
uint   u_in=-401280;
```

Hexadecimal: numbers 0-9, the letters a-f or A-F for the values of 10-15; start with 0x or 0X.

Examples:

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

See also

Typecasting

Character Constants

Characters as elements of a string in MQL4 are indexes in the Unicode character set. They are hexadecimal values that can be cast into integers, and that can be manipulated by integer operations like addition and subtraction.

Any single character in quotation marks or a hexadecimal ASCII code of a character as '\x10' is a character constant and is of ushort type. For example, a record of '0' type is a numerical value 30, that corresponds to the index of zero in the table of characters.

Example:

```
void OnStart()   {
//--- define character constants
   int symbol_0='0';
   int symbol_9=symbol_0+9; // get symbol '9'
//--- output values of constants
   printf("In a decimal form: symbol_0 = %d,  symbol_9 = %d",symbol_0,symb
   printf("In a hexadecimal form: symbol_0 = 0x%x,  symbol_9 = 0x%x",symbo
//--- enter constants into a string
   string test="";
   StringSetCharacter(test,0,symbol_0);
   StringSetCharacter(test,1,symbol_9);
//--- this is what they look like in a string
   Print(test);
  }
```

A backslash is a control character for a compiler when dealing with constant strings and character constants in a source text of a program. Some symbols, for example a single quote ('), double quotes ("), backslash (\) and control characters can be represented as a combination of symbols that start with a backslash (\), according to the below table:

Character name	Mnemonic code or image	Record in MQL4	Numeric value
new line (line feed)	LF	'\n'	10
horizontal tab	HT	'\t'	9
carriage return	CR	'\r'	13
backslash	\	'\\'	92

single quote	'	'\''	39
double quote	"	'\"'	34
hexadecimal code	hhhh	'\xhhhh'	1 to 4 hexadecimal characters
decimal code	d	'\d'	decimal number from 0 to 65535

If a backslash is followed by a character other than those described above, result is undefined.

Example

```
void OnStart()
  {
//--- declare character constants
   int a='A';
   int b='$';
   int c='©';      // code 0xA9
   int d='\xAE';   // code of the symbol ®
//--- output print constants
   Print(a,b,c,d);
//--- add a character to the string
   string test="";
   StringSetCharacter(test,0,a);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,b);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,c);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,d);
   Print(test);
//--- represent characters as a number
   int a1=65;
   int b1=36;
   int c1=169;
   int d1=174;
//--- add a character to the string
   StringSetCharacter(test,1,a1);
   Print(test);
//--- add a character to the string
   StringSetCharacter(test,1,b1);
   Print(test);
//--- add a character to the string
   StringSetCharacter(test,1,c1);
   Print(test);
//--- add a character to the string
   StringSetCharacter(test,1,d1);
   Print(test);
  }
```

As it was mentioned above, the value of a character constant (or variable) is an index in the table of characters. Index being an integer, it can be written in different ways.

```
void OnStart()
  {
//---
   int a=0xAE;     // the code of ® corresponds to the '\xAE' literal
   int b=0x24;     // the code of $ corresponds to the '\x24' literal
   int c=0xA9;     // the code of © corresponds to the '\xA9' literal
   int d=0x263A;   // the code of ☺ corresponds to the '\x263A' literal
//--- show values
   Print(a,b,c,d);
//--- add a character to the string
   string test="";
   StringSetCharacter(test,0,a);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,b);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,c);
   Print(test);
//--- replace a character in a string
   StringSetCharacter(test,0,d);
   Print(test);
//--- codes of suits
   int a1=0x2660;
   int b1=0x2661;
   int c1=0x2662;
   int d1=0x2663;
//--- add a character of spades
   StringSetCharacter(test,1,a1);
   Print(test);
//--- add a character of hearts
   StringSetCharacter(test,2,b1);
   Print(test);
//--- add a character of diamonds
   StringSetCharacter(test,3,c1);
   Print(test);
//--- add a character of clubs
   StringSetCharacter(test,4,d1);
   Print(test);
//--- Example of character literals in a string
   test="Queen\x2660Ace\x2662";
   printf("%s",test);
  }
```

The internal representation of a character literal is the ushort type. Character constants can accept values from 0 to 65535.

See also

StringSetCharacter(), StringGetCharacter(), ShortToString(), ShortArrayToString(), StringToShortArray()

Datetime Type

The datetime type is intended for storing the date and time as the number of seconds elapsed since January 01, 1970. This type occupies 8 bytes of memory.

Constants of the date and time can be represented as a literal string, which consists of 6 parts showing the numerical value of the year, month, day (or day, month, year), hours, minutes and seconds. The constant is enclosed in single quotation marks and starts with the D character.

Values range from 1 January, 1970 to 31 December, 3000. Either date (year , month, day) or time (hours, minutes, seconds), or all together can be omitted.

With literal date specification, it is desirable that you specify year, month and day. Otherwise the compiler returns a warning about an incomplete entry.

Examples:

```
datetime NY=D'2015.01.01 00:00';      // Time of beginning of year 2015 date
datetime d2=D'19.07.1980 12:30:27';   // Equal to D'1980.07.19 12:30:27';
datetime d3=D'19.07.1980 12';         // Equal to D'1980.07.19 12:00:00'
datetime d4=D'01.01.2004';            // Equal to D'01.01.2004 00:00:00'
datetime compilation_date=__DATE__;             // Compilation date
datetime compilation_date_time=__DATETIME__;    // Compilation date and ti
datetime compilation_time=__DATETIME__-__DATE__;// Compilation time
//--- Examples of declarations after which compiler warnings will be retur
datetime warning1=D'12:30:27';        // Equal to D'[date of compilation] 1
datetime warning2=D'';                // Equal to __DATETIME__
```

The string representation of datetime type depends on compilation mode:

```
  datetime date=D'2014.03.05 15:46:58';
  string str="mydate="+date;
//--- str="mydate=1394034418" - without #property strict
//--- str="mydate=2014.03.05 15:46:58" - with #property strict
```

See also

Structure of the Date Type, Date and Time, TimeToString, StringToTime

Color Type

The color type is intended for storing information about color and occupies 4 bytes in memory. The first byte is ignored, the remaining 3 bytes contain the RGB-components.

Color constants can be represented in three ways: literally, by integers, or by name (for named Web-colors only).

Literal representation consists of three parts representing numerical rate values of the three main color components: red, green, blue. The constant starts with 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 the RGB. They represent a decimal value of the hexadecimal integer representation.

Specific colors reflect the so-called Web-colors set.

Examples:

```
//--- Literals C'128,128,128'    // Gray
C'0x00,0x00,0xFF' // Blue
//color names
clrRed               // Red
clrYellow            // Yellow
clrBlack             // Black
//--- Integral representations
0xFFFFFF          // White
16777215          // White
0x008000          // Green
32768             // Green
```

See also

Web Colors, ColorToString, StringToColor, Typecasting

Bool Type

The bool type is intended to store the logical values of true or false, numeric representation of them is 1 or 0, respectively.

Examples:

```
bool a = true; bool b = false;
bool c = 1;
```

The internal representation is a whole number 1 byte large. It should be noted that in logical expressions you can use other integer or real types or expressions of these types - the compiler will not generate any error. In this case, the zero value will be interpreted as false, and all other values - as true.

Examples:

```
   int i=5;
   double d=-2.5;
   if(i) Print("i = ",i," and is set to true");
   else Print("i = ",i," and is set to false");

   if(d) Print("d = ",d," and has the true value");
   else Print("d = ",d," and has the false value");

   i=0;
   if(i) Print("i = ",i," and has the true value");
   else Print("i = ",i," and has the false value");

   d=0.0;
   if(d) Print("d = ",d," and has the true value");
   else Print("d = ",d," and has the false value");

//--- Execution results
//   i= 5 and has the true value
//   d= -2.5 and has the true value
//   i= 0 and has the false value
//   d= 0 and has the false value
```

See also

Boolean Operations, Precedence Rules

Enumerations

Data of the enum type belong to a certain limited set of data. Defining the enumeration type:

```
enum name of enumerable type  {
   list of values
  };
```

The list of values is a list of identifiers of named constants separated by commas.

Example:

```
enum months  // enumeration of named constants
   {
    January,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
   };
```

After the enumeration is declared, a new integer-valued 4-byte data type appears. Declaration of the new data type allows the compiler to strictly control types of passed parameters, because enumeration introduces new named constants. In the above example, the January named constant has the value of 0, February - 1, December - 11.

Rule: If a certain value is not assigned to a named constant that is a member of the enumeration, its new value will be formed automatically. If it is the first member of the enumeration, the 0 value will be assigned to it. For all subsequent members, values will be calculated based on the value of the previous members by adding one.

Example:

```
enum intervals  // Enumeration of named constants
   {
    month=1,     // Interval of one month
    two_months,  // Two months
    quarter,     // Three months - quarter
    halfyear=6,  // Half a year
    year=12,     // Year - 12 months
   };
```

Notes

- Unlike C++, the size of the internal representation of the enumerated type in MQL4 is always equal to 4 bytes. That is, sizeof(months) returns the value 4.
- Unlike C++, an anonymous enumeration can't be declared in MQL4. That is, a unique name must be always specified after the enum keyword.

See also

Typecasting

Real Types (double, float)

Real types (or floating-point types) represent values with a fractional part. In the MQL4 language there are two types for floating point numbers.The method of representation of real numbers in the computer memory is defined by the IEEE 754 standard and is independent of platforms, operating systems or programming languages.

Type	Size in bytes	Minimal Positive Value	Maximum Value	C++ Analog
float	4	1.175494351e-38	3.402823466e+38	float
double	8	2.2250738585072014e-308	1.7976931348623158e+308	double

The double name means that the accuracy of these numbers is twice the accuracy of the float type numbers. In most cases, the double type is the most convenient one. In many cases the limited precision of float numbers is not enough. The reason why the float type is still used is saving the memory (this is important for large arrays of real numbers).

Floating-point constants consist of an integer part, a point (.) and the fractional part. The integer and fractional parts are sequences of decimal digits.

Examples:

```
   double a=12.111;    double b=-956.1007;
   float  c =0.0001;
   float  d =16;
```

There is a scientific way of writing real constants, often this method of recording is more compact than the traditional one.

Example:

```
   double c1=1.12123515e-25;
   double c2=0.000000000000000000000000112123515; // 24 zero after the dec

   Print("1. c1 =",DoubleToString(c1,16));
   // Result: 1. c1 = 0.0000000000000000

   Print("2. c1 =",DoubleToString(c1,-16));
   // Result: 2. c1 = 1.1212351499999999e-025

   Print("3. c2 =",DoubleToString(c2,-16));
   // Result: 3. c2 = 1.1212351499999999e-025
```

It should be remembered that real numbers are stored in memory with some limited accuracy in the binary system, while generally the decimal notation is used. That's why many numbers that are precisely represented in the decimal system can be written only as an infinite fraction in the binary system.

For example, numbers 0.3 and 0.7 are represented in the computer as infinite fractions, while the number of 0.25 is stored exactly, because it represents the power of two.

In this regard, it is strongly recommended not to compare two real numbers for equality, because such a comparison is not correct.

Example:

```
void OnStart()
  {
//---
   double three=3.0;
   double x,y,z;
   x=1/three;
   y=4/three;
   z=5/three;
   if(x+y==z) Print("1/3 + 4/3 == 5/3");
   else Print("1/3 + 4/3 != 5/3");
// Result: 1/3 + 4/3 != 5/3
  }
```

If you still need to compare the equality of two real numbers, then you can do this in two different ways. The first way is to compare the difference between two numbers with some small quantity that specifies the accuracy of comparison.

Example:

```
bool EqualDoubles(double d1,double d2,double epsilon)
  {
   if(epsilon<0) epsilon=-epsilon;
//---
   if(d1-d2>epsilon) return false;
   if(d1-d2<-epsilon) return false;
//---
   return true;
  }
void OnStart()
  {
   double d_val=0.7;
   float  f_val=0.7;
   if(EqualDoubles(d_val,f_val,0.000000000000001)) Print(d_val," equals ",
   else Print("Different: d_val = ",DoubleToString(d_val,16),
              "  f_val = ",DoubleToString(f_val,16));
// Result: Different: d_val= 0.7000000000000000   f_val= 0.6999999880079071
  }
```

Note that the value of epsilon in the above example can not be less than the predefined constant DBL_EPSILON. The value of this constant is 2.2204460492503131e-016. The constant corresponding to the float type is FLT_EPSILON = 1.192092896e-07. The meaning of these values is the following: it is the lowest value that satisfies the condition 1.0 + DBL_EPSILON! = 1.0 (for numbers of float type 1.0 + FLT_EPSILON! = 1.0).

The second way offers comparing the normalized difference of two real numbers with zero. It's meaningless to compare the difference of normalized numbers with a zero, because any mathematical operation with normalized numbers gives a non-normalized result.

Example:

```
bool CompareDoubles(double number1,double number2)
  {
   if(NormalizeDouble(number1-number2,8)==0) return(true);
   else return(false);
  }
void OnStart()
  {
   double d_val=0.3;
   float  f_val=0.3;
   if(CompareDoubles(d_val,f_val)) Print(d_val," equals ",f_val);
   else Print("Different: d_val = ",DoubleToString(d_val,16),
              "  f_val = ",DoubleToString(f_val,16));
// Result: Different: d_val= 0.3000000000000000   f_val= 0.3000000119209290
  }
```

Some operations of the mathematical co-processor can result in the invalid real number, which can't be used in mathematical operations and operations of comparison, because the result of operations with invalid real numbers is undefined. For example, when trying to calculate the arcsine of 2, the result is the negative infinity.

Example:

```
   double abnormal = MathArcsin(2.0);
   Print("MathArcsin(2.0) =",abnormal);
// Result:  MathArcsin(2.0) = -1.#IND
```

Besides the minus infinity there is the plus infinity and NaN (not a number). To determine that this number is invalid, you can use MathIsValidNumber(). According to the IEEE standard, they have a special machine representation. For example, plus infinity for the double type has the bit representation of 0x7FF0 0000 0000 0000.

Examples:

```
struct str1
  {
   double d;
  };
struct str2
  {
   long l;
  };

//--- Start
   str1 s1;
   str2 s2;
//---
   s1.d=MathArcsin(2.0);        // Get the invalid number -1.#IND
   s2=s1;
   printf("1.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0xFFFF000000000000;     // invalid number -1.#QNAN
   s1=s2;
   printf("2.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF7000000000000;     // greatest non-number SNaN
   s1=s2;
   printf("3.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF8000000000000;     // smallest non-number QNaN
   s1=s2;
   printf("4.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FFF000000000000;     // greatest non-number QNaN
   s1=s2;
   printf("5.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF0000000000000;     // Positive infinity 1.#INF and smallest r
   s1=s2;
   printf("6.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0xFFF0000000000000;     // Negative infinity -1.#INF
   s1=s2;
   printf("7.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x8000000000000000;     // Negative zero -0.0
   s1=s2;
   printf("8.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x3FE0000000000000;     // 0.5
   s1=s2;
   printf("9.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x3FF0000000000000;     // 1.0
   s1=s2;
   printf("10.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FEFFFFFFFFFFFFF;     // Greatest normalized number (MAX_DBL)
   s1=s2;
   printf("11.  %.16e %I64X",s1.d,s2.l);
//---
   s2.l=0x0010000000000000;     // Smallest positive normalized (MIN_DBL)
   s1=s2;
   printf("12.  %.16e %.16I64X",s1.d,s2.l);
//---
   s1.d=0.7;                    // Show that the number of 0.7 - endless f
   s2=s1;
   printf("13.  %.16e %.16I64X",s1.d,s2.l);
/*
1.  -1.#IND00 FFF8000000000000
2.  -1.#QNAN0 FFFF000000000000
3.   1.#SNAN0 7FF7000000000000
4.   1.#QNAN0 7FF8000000000000
5.   1.#QNAN0 7FFF000000000000
6.   1.#INF00 7FF0000000000000
7.  -1.#INF00 FFF0000000000000
8.  -0.000000 8000000000000000
9.   0.500000 3FE0000000000000
```

```
10.  1.000000 3FF0000000000000
11.  1.7976931348623157e+308 7FEFFFFFFFFFFFFF
12.  2.2250738585072014e-308 0010000000000000
13.  6.9999999999999996e-001 3FE6666666666666
*/
```

See also

DoubleToString, NormalizeDouble, Numeric Type Constants

String Type

The string type is used for storing text strings. A text string is a sequence of characters in the Unicode format with the final zero at the end of it. A string constant can be assigned to a string variable. A string constant is a sequence of Unicode characters enclosed in double quotes: "This is a string constant".

If you need to include a double quote (") into a string, the backslash character (\) must be put before it. Any special character constants can be written in a string, if the backslash character (\) is typed before them.

Examples:

```
string svar="This is a character string"; string svar2=StringSubstr(svar,0
Print("Copyright symbol\t\x00A9");
FileWrite(handle,"This string contains a new line symbols \n");
string MT4path="C:\\Program Files\\MetaTrader 4";
```

To make the source code readable, long constant strings can be split into parts without addition operation. During compilation, these parts will be combined into one long string:

```
//--- Declare a long constant string
   string HTML_head="<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transit
                    " \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitiona
                    "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n"
                    "<head>\n"
                    "<meta http-equiv=\"Content-Type\" content=\"text/html
                    "<title>Trade Operations Report</title>\n"
                    "</head>";
//--- Output the constant string into log
   Print(HTML_head);
  }
```

Internal representation of the string type is a structure of 12 bytes long:

```
#pragma pack(push,1)
struct MqlString
  {
   int      size;       // 32-bit integer, contains size of the buffer, al
   LPWSTR   buffer;     // 32-bit address of the buffer, containing the st
   int      reserved;   // 32-bit integer, reserved.
  };
#pragma pack(pop,1)
```

See also

Conversion Functions, String Functions, FileOpen(), FileReadString(), FileWriteString()

Structures and Classes

Structures

A structure is a set of elements of any type (except for the void type). Thus, the structure combines logically related data of different types.

Structure Declaration

The structure data type is determined by the following description:

```
struct structure_name   {
   elements_description
  };
```

The structure name can't be used as an identifier (name of a variable or function). It should be noted that in MQL4 structure elements follow one another directly, without alignment. In C++ such an order is made to the compiler using the following instruction:

```
#pragma pack(1)
```

If you want to do another alignment in the structure, use auxiliary members, "fillers" to the right size.

Example:

```
struct trade_settings
  {
   uchar  slippage;     // value of the permissible slippage-size 1 byte
   char   reserved1;    // skip 1 byte
   short  reserved2;    // skip 2 bytes
   int    reserved4;    // another 4 bytes are skipped. ensure alignment c
   double take;         // values of the price of profit fixing
   double stop;         // price value of the protective stop
  };
```

Such a description of aligned structures is necessary only for transferring to imported dll-functions.

Attention: This example illustrates incorrectly designed data. It would be better first to declare the *take* and *stop* large data of the double type, and then declare the *slippage* member of the uchar type. In this case, the internal representation of data will always be the same regardless of the value specified in *#pragma pack()*.

If a structure contains variables of the string type and/or object of a dynamic array, the compiler assigns an implicit constructor to such a structure. This constructor resets all the structure members of string type and correctly initializes objects of the dynamic array.

Simple Structures

Structures that do not contain strings or objects of dynamic arrays are called simple structures; variables of such structures can be freely copied to each other, even if they are different structures. Variables of simple structures, as well as their array can be passed as parameters to functions imported from DLL.

Access to Structure Members

The name of a structure becomes a new data type, so you can declare variables of this type. The structure can be declared only once within a project. The structure members are accessed using the point operation (.).

Example:

```
struct trade_settings
  {
   double take;         // values of the profit fixing price
   double stop;         // value of the protective stop price
   uchar  slippage;     // value of the acceptable slippage
  };
//--- create up and initialize a variable of the trade_settings type
trade_settings my_set={0.0,0.0,5};
if (input_TP>0) my_set.take=input_TP;
```

Classes

Classes differ from structures in the following:

- the keyword class is used in declaration;
- by default, all class members have access specifier private, unless otherwise indicated. Data-members of the structure have the default type of access as public, unless otherwise indicated;
- class objects always have a table of virtual functions, even if there are no virtual functions declared in the class. Structures can't have virtual functions;
- the new operator can be applied to class objects; this operator cannot be applied to structures;
- classes can be inherited only from classes, structures can be inherited only from structures.

Classes and structures can have an explicit constructor and destructor. If your constructor is explicitly defined, the initialization of a structure or class variable using the initializing sequence is impossible.

Example:

```
struct trade_settings
  {
   double take;         // values of the profit fixing price
   double stop;         // value of the protective stop price
   uchar  slippage;     // value of the acceptable slippage
   //--- Constructor
          trade_settings() { take=0.0; stop=0.0; slippage=5; }
   //--- Destructor
         ~trade_settings() { Print("This is the end"); }
  };
//--- Compiler will generate an error message that initialization is impos
trade_settings my_set={0.0,0.0,5};
```

Constructors and Destructors

A constructor is a special function, which is called automatically when creating an object of a structure or class and is usually used to initialize class members. Further we will talk only about classes, while the same applies to structures, unless otherwise indicated. The name of a constructor must match the class name. The constructor has no return type (you can specify the void type).

Defined class members strings, dynamic arrays and objects that require initialization will be in any case initialized, regardless of whether there is a constructor.

Each class can have multiple constructors, differing by the number of parameters and the initialization list. A constructor that requires specifying parameters is called a parametric constructor.

A constructor with no parameters is called **a default constructor**. If no constructors are declared in a class, the compiler creates a default constructor during compilation.

```
//+------------------------------------------------------------------+
//| Class for working with a date                                    |
//+------------------------------------------------------------------+
class MyDateClass
  {
private:
   int               m_year;          // Year
   int               m_month;         // Month
   int               m_day;           // Day of the month
   int               m_hour;          // Hour in a day
   int               m_minute;        // Minutes
   int               m_second;        // Seconds
public:
   //--- Default constructor
                     MyDateClass(void);
   //--- Parametric constructor
                     MyDateClass(int h,int m,int s);
  };
```

A constructor can be declared in the class description and then its body can be defined. For example, two constructors of MyDateClass can be defined the following way:

```
//+------------------------------------------------------------------+
//| Default constructor                                              |
//+------------------------------------------------------------------+
MyDateClass::MyDateClass(void)
  {
//---
   MqlDateTime mdt;
   datetime t=TimeCurrent(mdt);
   m_year=mdt.year;
   m_month=mdt.mon;
   m_day=mdt.day;
   m_hour=mdt.hour;
   m_minute=mdt.min;
   m_second=mdt.sec;
   Print(__FUNCTION__);
  }
//+------------------------------------------------------------------+
//| Parametric constructor                                           |
//+------------------------------------------------------------------+
MyDateClass::MyDateClass(int h,int m,int s)
  {
   MqlDateTime mdt;
   datetime t=TimeCurrent(mdt);
   m_year=mdt.year;
   m_month=mdt.mon;
   m_day=mdt.day;
   m_hour=h;
   m_minute=m;
   m_second=s;
   Print(__FUNCTION__);
  }
```

In the default constructor, all members of the class are filled using the TimeCurrent() function. In the parametric constructor only hour values are filled in. Other members of the class (m_year, m_month and m_day) will be automatically initialized with the current date.

The default constructor has a special purpose when initializing an array of objects of its class. The constructor, all parameters of which have default values, **is not** a default constructor. Here is an example:

```
//+------------------------------------------------------------------+
//| Class with a default constructor                                 |
//+------------------------------------------------------------------+
class CFoo
  {
   datetime          m_call_time;     // Time of the last object call
```

```
public:
   //--- Constructor with a parameter that has a default value is not a de
                     CFoo(const datetime t=0){m_call_time=t;};
   //--- Copy constructor
                     CFoo(const CFoo &foo){m_call_time=foo.m_call_time;};

   string ToString(){return(TimeToString(m_call_time,TIME_DATE|TIME_SECOND
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
// CFoo foo; // This variant cannot be used - a default constructor is not
//--- Possible options to create the CFoo object
   CFoo foo1(TimeCurrent());     // An explicit call of a parametric const
   CFoo foo2();                  // An explicit call of a parametric const
   CFoo foo3=D'2009.09.09';      // An implicit call of a parametric const
   CFoo foo40(foo1);             // An explicit call of a copy constructor
   CFoo foo41=foo1;              // An implicit call of a copy constructor
   CFoo foo5;                    // An explicit call of a default construc
                                 // then a parametric constructor with a d
//--- Possible options to receive CFoo pointers
   CFoo *pfoo6=new CFoo();       // Dynamic creation of an object and rece
   CFoo *pfoo7=new CFoo(TimeCurrent());// Another option of dynamic object
   CFoo *pfoo8=GetPointer(foo1); // Now pfoo8 points to object foo1
   CFoo *pfoo9=pfoo7;            // pfoo9 and pfoo7 point to one and the s
   // CFoo foo_array[3];         // This option cannot be used - a default
//--- Show the value of m_call_time
   Print("foo1.m_call_time=",foo1.ToString());
   Print("foo2.m_call_time=",foo2.ToString());
   Print("foo3.m_call_time=",foo3.ToString());
   Print("foo4.m_call_time=",foo4.ToString());
   Print("foo5.m_call_time=",foo5.ToString());
   Print("pfoo6.m_call_time=",pfoo6.ToString());
   Print("pfoo7.m_call_time=",pfoo7.ToString());
   Print("pfoo8.m_call_time=",pfoo8.ToString());
   Print("pfoo9.m_call_time=",pfoo9.ToString());
//--- Delete dynamically created arrays
   delete pfoo6;
   delete pfoo7;
   //delete pfoo8;  // You do not need to delete pfoo8 explicitly, since i
   //delete pfoo9;  // You do not need to delete pfoo9 explicitly. since i
  }
```

If you uncomment these strings

```
   //CFoo foo_array[3];     // This variant cannot be used - a default cons
```

or

```
   //CFoo foo_dyn_array[];  // This variant cannot be used - a default cons
```

then the compiler will return an error for them "default constructor is not defined".

If a class has a user-defined constructor, the default constructor is not generated by the compiler. This means that if a parametric constructor is declared in a class, but a default constructor is not declared, you can not declare the arrays of objects of this class. The compiler will return an error for this script:

```
//+------------------------------------------------------------------+
//| Class without a default constructor                              |
//+------------------------------------------------------------------+
class CFoo
  {
   string            m_name;
public:
                     CFoo(string name) { m_name=name;}
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Get the "default constructor is not defined" error during compilatio
   CFoo badFoo[5];
  }
```

In this example, the CFoo class has a declared parametric constructor - in such cases, the compiler does not create a default constructor automatically during compilation. At the same time when you declare an array of objects, it is assumed that all objects should be created and initialized automatically. During auto-initialization of an object, it is necessary to call a default constructor, but since the default constructor is not explicitly declared and not automatically generated by the compiler, it is impossible to create such an object. For this reason, the compiler generates an error at the compilation stage.

There is a special syntax to initialize an object using a constructor. Constructor initializers (special constructions for initialization) for the members of a struct or class can be specified in the initialization list.

An initialization list is a list of initializers separated by commas, which comes after the colon after the list of parameters of a constructor and precedes the body (goes before an opening brace). There are several requirements:

- Initialization lists can be used only in constructors;
- Parent members cannot be initialized in the initialization list;
- The initialization list must be followed by a definition (implementation) of a function.

Here is an example of several constructors for initializing class members.

```
//+------------------------------------------------------------------+
//| Class for storing the name of a character                        |
//+------------------------------------------------------------------+
class CPerson
  {
   string            m_first_name;     // First name
   string            m_second_name;    // Second name
public:
   //--- An empty default constructor
                     CPerson() {Print(__FUNCTION__);};
   //--- A parametric constructor
                     CPerson(string full_name);
   //--- A constructor with an initialization list
                     CPerson(string surname,string name): m_second_name(su
   void PrintName(){PrintFormat("Name=%s Surname=%s",m_first_name,m_second
  };
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
CPerson::CPerson(string full_name)
  {
   int pos=StringFind(full_name," ");
   if(pos>=0)
     {
      m_first_name=StringSubstr(full_name,0,pos);
      m_second_name=StringSubstr(full_name,pos+1);
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Get an error "default constructor is not defined"
   CPerson people[5];
   CPerson Tom="Tom Sawyer";                       // Tom Sawyer
   CPerson Huck("Huckleberry","Finn");             // Huckleberry Finn
   CPerson *Pooh = new CPerson("Winnie","Pooh");  // Winnie the Pooh
   //--- Output values
   Tom.PrintName();
   Huck.PrintName();
   Pooh.PrintName();

   //--- Delete a dynamically created object
   delete Pooh;
  }
```

In this case, the CPerson class has three constructors:

1. An explicit default constructor, which allows creating an array of objects of this class;
2. A constructor with one parameter, which gets a full name as a parameter and divides it to the name and second name according to the found space;
3. A constructor with two parameters that contains an initialization list. Initializers - m_second_name(surname) and m_first_name(name).

Note that the initialization using a list has replaced an assignment. Individual members must be initialized as:

```
class_member (a list of expressions)
```

In the initialization list, members can go in any order, but all members of the class will be initialized according to the order of their announcement. This means that in the third constructor, first the m_first_name member will be initialized, as it is announced first, and only after it m_second_name is initialized. This should be taken into account in cases where the initialization of some members of the class depends on the values in other class members.

If a default constructor is not declared in the base class, and at the same time one or more constructors with parameters are declared, you should always call one of the base class constructors in the initialization list. It goes through the comma as ordinary members of the list and will be called first during object initialization, no matter where in the initialization list it is located.

```
//+------------------------------------------------------------------+
//| Base class                                                       |
//+------------------------------------------------------------------+
class CFoo
  {
   string            m_name;
public:
   //--- A constructor with an initialization list
                     CFoo(string name) : m_name(name) { Print(m_name);}
  };
//+------------------------------------------------------------------+
//| Class derived from CFoo                                          |
//+------------------------------------------------------------------+
class CBar : CFoo
  {
   CFoo              m_member;      // A class member is an object of the
public:
   //--- A default constructor in the initialization list calls the constr
                     CBar(): m_member(_Symbol), CFoo("CBAR") {Print(__FUNC
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   CBar bar;
  }
```

In this example, when creating the bar object, a default constructor CBar() will be called, in which first a constructor for the parent CFoo is called, and then comes a constructor for the m_member class member.

A destructor is a special function that is called automatically when a class object is destroyed. The name of the destructor is written as a class name with a tilde (~). Strings, dynamic arrays and objects, requiring deinitialization, will be de-initialized anyway, regardless of the destructor presence or absence. If there is a destructor, these actions will be performed after calling the destructor.

Destructors are always virtual, regardless of whether they are declared with the virtual keyword or not.

Defining Class Methods

Class function-methods can be defined both inside the class and outside the class declaration. If the method is defined within a class, then its body comes right after the method declaration.

Example:

```
class CTetrisShape
  {
protected:
   int               m_type;
   int               m_xpos;
   int               m_ypos;
   int               m_xsize;
   int               m_ysize;
   int               m_prev_turn;
   int               m_turn;
   int               m_right_border;
public:
   void              CTetrisShape();
   void              SetRightBorder(int border) { m_right_border=border; }
   void              SetYPos(int ypos)          { m_ypos=ypos;           }
   void              SetXPos(int xpos)          { m_xpos=xpos;           }
   int               GetYPos()                  { return(m_ypos);        }
   int               GetXPos()                  { return(m_xpos);        }
   int               GetYSize()                 { return(m_ysize);       }
   int               GetXSize()                 { return(m_xsize);       }
   int               GetType()                  { return(m_type);        }
   void              Left()                     { m_xpos-=SHAPE_SIZE;    }
   void              Right()                    { m_xpos+=SHAPE_SIZE;    }
   void              Rotate()                   { m_prev_turn=m_turn; if(+
   virtual void      Draw()                     { return;                }
   virtual bool      CheckDown(int& pad_array[]);
   virtual bool      CheckLeft(int& side_row[]);
   virtual bool      CheckRight(int& side_row[]);
  };
```

Functions from SetRightBorder(int border) to Draw() are declared and defined directly inside the CTetrisShape class.

The CTetrisShape() constructor and methods CheckDown(int& pad_array[]), CheckLeft(int& side_row[]) and CheckRight(int& side_row[]) are only declared inside the class, but not defined yet. Definitions of these functions will be further in the code. In order to define the method outside the class, the scope resolution operator is used, the class name is used as the scope.

Example:

```
//+------------------------------------------------------------------+
//| Constructor of the basic class                                   |
//+------------------------------------------------------------------+
void CTetrisShape::CTetrisShape()
  {
   m_type=0;
   m_ypos=0;
   m_xpos=0;
   m_xsize=SHAPE_SIZE;
   m_ysize=SHAPE_SIZE;
   m_prev_turn=0;
   m_turn=0;
   m_right_border=0;
  }
//+------------------------------------------------------------------+
//| Checking ability to move down (for the stick and cube)           |
//+------------------------------------------------------------------+
bool CTetrisShape::CheckDown(int& pad_array[])
  {
   int i,xsize=m_xsize/SHAPE_SIZE;
//---
   for(i=0; i<xsize; i++)
     {
      if(m_ypos+m_ysize>=pad_array[i]) return(false);
     }
//---
   return(true);
  }
```

Public, Protected and Private Access Modifiers

When developing a new class, it is recommended to restrict access to the members from the outside. For this purpose keywords private or protected are used. In this case, hidden data can be accessed only from function-methods of the same class. If the *protected* keyword is used, hidden data can be accessed also from methods of classes - inheritors of this class. The same method can be used to restrict the access to functions-methods of a class.

If you need to completely open access to members and/or methods of a class, use the keyword public.

Example:

```
class CTetrisField
  {
private:
   int                  m_score;                            // Score
   int                  m_ypos;                             // Current positi
   int                  m_field[FIELD_HEIGHT][FIELD_WIDTH]; // Matrix of the
   int                  m_rows[FIELD_HEIGHT];               // Numbering of t
   int                  m_last_row;                         // Last free row
   CTetrisShape        *m_shape;                            // Tetris figure
   bool                 m_bover;                            // Game over
public:
   void                 CTetrisField() { m_shape=NULL; m_bover=false; }
   void                 Init();
   void                 Deinit();
   void                 Down();
   void                 Left();
   void                 Right();
   void                 Rotate();
   void                 Drop();
private:
   void                 NewShape();
   void                 CheckAndDeleteRows();
   void                 LabelOver();
  };
```

Any class members and methods declared after the specifier **public:** (and before the next access specifier) are available in any reference to the class object by the program. In this example these are the following members: functions CTetrisField(), Init(), Deinit(), Down(), Left(), Right(), Rotate() and Drop().

Any members that are declared after the access specifier to the elements **private:** (and before the next access specifier) are available only to members-functions of this class. Specifiers of access to elements always end with a colon (:) and can appear in the class definition many times.

Access to the members of the basis class can be redefined during inheritance in derived classes.

See also

Object-Oriented Programming

Dynamic Array Object

Dynamic Arrays

Maximum 4-dimension array can be declared. When declaring a dynamic array (an array of unspecified value in the first pair of square brackets), the compiler automatically creates a variable of the above structure (a dynamic array object) and provides a code for the correct initialization.

Dynamic arrays are automatically freed when going beyond the visibility area of the block they are declared in.

Example:

```
double matrix[][10][20]; // 3-dimensional dynamic array ArrayResize(matrix
```

Static Arrays

When all significant array dimensions are explicitly specified, the compiler pre-allocates the necessary memory size. Such an array is called static. Nevertheless, the compiler allocates additional memory for the object of a dynamic array, which (object) is associated with the pre-allocated static buffer (memory part for storing the array).

Creating a dynamic array object is due to the possible need to pass this static array as a parameter to some function.

Examples:

```
double stat_array[5]; // 1-dimensional static array
some_function(stat_array);
...
bool some_function(double& array[])
  {
   if(ArrayResize(array,100)<0) return(false);
   ...
   return(true);
  }
```

Arrays in Structures

When a static array is declared as a member of a structure, a dynamic array object is not created. This is done to ensure compatibility of data structures used in the Windows API.

However, static arrays that are declared as members of structures can also be passed to MQL5 functions. In this case, when passing the parameter, a temporary object of a dynamic array will be created. Such an object is linked with the static array - member of structure.

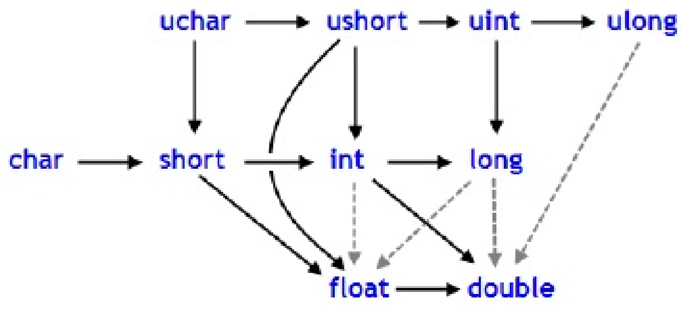
See also

Array Functions, Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Typecasting

Casting Numeric Types

Often a necessity occurs to convert one numeric type into another. Not all numeric types can be converted into another. Here is the scheme of allowed casting:



Solid lines with arrows indicate changes that are performed almost without any loss of information. Instead of the char type, the bool type can be used (both take 1 byte of memory), instead of type int, the color type can be used (4 bytes), instead of the long type, datetime can be used (take 8 bytes). The four dashed grey lines, also arrowed, denote conversions, when the loss of precision can occur. For example, the number of digits in an integer equal to 123456789 (int) is higher than the number of digits that can be represented by float.

```
   int n=123456789;    float f=n;    // the content of f is equal to 1.234
   Print("n = ",n,"   f = ",f);
   // result n= 123456789    f= 123456792.00000
```

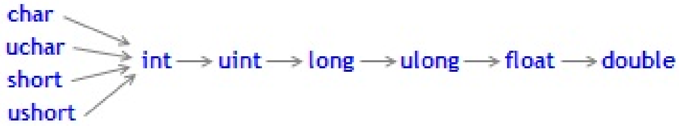
A number converted into float has the same order, but is less accurate. Conversions, contrary to black arrows, can be performed with possible data loss. Conversions between char and uchar, short and ushort, int and uint, long and ulong (conversions to both sides), may lead to the loss of data.

As a result of converting floating point values to integer type, the fractional part is always deleted. If you want to round off a float to the nearest whole number (which in many cases is more useful), you should use MathRound().

Example:

```
//--- Gravitational acceleration
   double g=9.8;
   double round_g=(int)g;
   double math_round_g=MathRound(g);
   Print("round_g = ",round_g);
   Print("math_round_g = ",math_round_g);
/*
   Result:
   round_g = 9
   math_round_g = 10
*/
```

If two values are combined by a binary operator, before the operation execution the operand of a lower type is converted to the higher type in accordance with the priority given in the below scheme:



The data types char, uchar, short, and ushort unconditionally are converted to the int type.

Examples:

```
   char   c1=3;
//--- First example
   double d2=c1/2+0.3;
   Print("c1/2 + 0.3 = ",d2);
// Result:   c1/2+0.3 = 1.3

//--- Second example
   d2=c1/2.0+0.3;
   Print("c1/2.0 + 0.3 = ",d2);
// Result:   c1/2.0+0.3 = 1.8
```

The calculated expression consists of two operations. In the first example, the variable c1 of the char type is converted to a temporary variable of the int type, because the second operand in the division operation, the constant 2, is of the higher type int. As a result of the integer division 3/2 we get the value 1, which is of the int type.

In the second operation of the first example, the second operand is the constant 0.3, which is of the double type, so the result of the first operation is converted into a temporary variable of the double type with a value of 1.0.

In the second example the variable of the char type c1 is converted to a temporary variable of the double type, because the second operand in the division operation, the constant 2.0, is of the double type; no further conversions are made.

Typecasting of Numeric Types

In the expressions of the MQL4 language both explicit and implicit typecasting can be used. The explicit typecasting is written as follows:

```
var_1 = (type)var_2;
```

An expression or function execution result can be used as the var_2 variable. The function style notation of the explicit typecasting is also possible:

```
var_1 = type(var_2);
```

Let's consider an explicit typecasting on the basis of the first example.

```
//--- Third example
   double d2=(double)c1/2+0.3;
   Print("(double)c1/2 + 0.3 = ",d2);
// Result:   (double)c1/2+0.3 = 1.80000000
```

Before the division operation is performed, the c1 variable is explicitly cast to the double type. Now the integer constant 2 is cast to the value 2.0 of the double type, because as a result of converting the first operand has taken the double type. In fact, the explicit typecasting is a unary operation.

Besides, when trying to cast types, the result may go beyond the permissible range. In this case, the truncation occurs. For example:

```
   char c;
   uchar u;
   c=400;
   u=400;
   Print("c = ",c); // Result c=-112
   Print("u = ",u); // Result u=144
```

Before operations (except for the assignment ones) are performed, the data are converted into the maximum priority type. Before assignment operations are performed, the data are cast into the target type.

Examples:

```
   int    i=1/2;         // no types casting, the result is 0
   Print("i = 1/2  ",i);

   int k=1/2.0;          // the expression is cast to the double type,
   Print("k = 1/2  ",k);  // then is to the target type of int, the result

   double d=1.0/2.0;     // no types casting, the result is 0.5
   Print("d = 1/2.0; ",d);

   double e=1/2.0;       // the expression is cast to the double type,
   Print("e = 1/2.0; ",e);// that is the same as the target type, the resu

   double x=1/2;         // the expression of the int type is cast to the c
   Print("x = 1/2; ",x);  // the result is 0.0
```

When converting long/ulong type into double, precision may be lost in case the integer value is greater than 9223372036854774784 or less than -9223372036854774784.

```
void OnStart()
  {
   long l_max=LONG_MAX;
   long l_min=LONG_MIN+1;
//--- define the highest integer value, which does not lose accuracy when
   while(l_max!=long((double)l_max))
      l_max--;
//--- define the lowest integer value, which does not lose accuracy when k
   while(l_min!=long((double)l_min))
      l_min++;
//--- derive the found interval for integer values
   PrintFormat("When casting an integer value to double, it must be "
               "within [%I64d, %I64d] interval",l_min,l_max);
//--- now, let's see what happens if the value falls out of this interval
   PrintFormat("l_max+1=%I64d, double(l_max+1)=%.f, ulong(double(l_max+1))
               l_max+1,double(l_max+1),long(double(l_max+1)));
   PrintFormat("l_min-1=%I64d, double(l_min-1)=%.f, ulong(double(l_min-1))
               l_min-1,double(l_min-1),long(double(l_min-1)));
//--- receive the following result
// When casting an integer value to double, it should be within [-92233720
// l_max+1=9223372036854774785, double(l_max+1)=9223372036854774800, ulong
// l_min-1=-9223372036854774785, double(l_min-1)=-9223372036854774800, ulo
  }
```

Typecasting for the String Type

The string type has the highest priority among simple types. Therefore, if one of operands of an operation is of the string type, the second operand will be cast to a string automatically. Note that for a string, a single dyadic two-place operation of addition is possible. The explicit casting of string to any numeric type is allowed.

Examples:

```
   string s1=1.0/8;            // the expression is cast to the double typ
   Print("s1 = 1.0/8; ",s1);     //   then is to the target type of string,
// result is "0.12500000" (a string containing 10 characters)

   string s2=NULL;             // string deinitialization
   Print("s2 = NULL; ",s2);      // the result is an empty string
   string s3="Ticket N"+12345; // the expression is cast to the string typ
   Print("s3 = \"Ticket N\"+12345",s3);

   string str1="true";
   string str2="0,255,0";
   string str3="2009.06.01";
   string str4="1.2345e2";
   Print(bool(str1));
   Print(color(str2));
   Print(datetime(str3));
   Print(double(str4));
```

Typecasting of Simple Structure Types

Data of the simple structures type can be assigned to each other only if all the members of both structures are of numeric types. In this case both operands of the assignment operation (left and right) must be of the structures type. The member-wise casting is not performed, a simple copying is done. If the structures are of different sizes, the number of bytes of the smaller size is copied. Thus the absence of union in MQL4 is compensated.

Examples:

```
struct str1
  {
   double  d;
  };
//---
struct str2
  {
   long    l;
  };
//---
struct str3
  {
   int     low_part;
   int     high_part;
  };
//---
struct str4
  {
   string  s;
  };
//+------------------------------------------------------------------+
void OnStart()
  {
   str1 s1;
   str2 s2;
   str3 s3;
   str4 s4;
//---
   s1.d=MathArcsin(2.0);        // get the invalid number -1. # IND
   s2=s1;
   printf("1.  %f    %I64X",s1.d,s2.l);
//---
   s3=s2;
   printf("2.  high part of long %.8X   low part of long %.8X",
          s3.high_part,s3.low_part);
//---
   s4.s="some constant string";
   s3=s4;
   printf("3.  buffer len is %d   constant string address is 0x%.8X",
          s3.low_part,s3.high_part);
  }
```

Another example illustrates the method of organizing a custom function for receiving RGB (Red, Green, Blue) representation from the color type. Create two structures of the same size but with different contents. For convenience, let's add a function returning the RGB representation of a color as a string.

```
#property script_show_inputs
input color          testColor=clrBlue;// set color for testing
//--- structure for representing color as RGB
struct RGB
  {
   uchar             blue;          // blue component of color
   uchar             green;         // green component of color
   uchar             red;           // red component of color
   uchar             empty;         // this byte is not used
   string            toString();    // function for receiving a string
  };
//--- function for showing color as a string
string RGB::toString(void)
  {
   string out="("+(string)red+":"+(string)green+":"+(string)blue+")";
   return out;
  }
//--- structure for storing of the built-in color type
struct builtColor
  {
   color             c;
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- a variable for storing in RGB
   RGB colorRGB;
//--- variable for storing the color type
   builtColor test;
   test.c=testColor;
//--- casting two structures by copying contents
   colorRGB=test;
   Print("color ",test.c,"=",colorRGB.toString());
//---
  }
```

Typecasting of Base Class Pointers to Pointers of Derivative Classes

Objects of the open generated class can also be viewed as objects of the corresponding base class. This leads to some interesting consequences. For example, despite the fact that objects of different classes, generated by a single base class, may differ significantly from each other, we can create a linked list (List) of them, as we view them as objects of the base type. But the converse is not true: the base class objects are not automatically objects of a derived class.

You can use the explicit casting to convert the base class pointers to the pointers of a derived class. But you must be fully confident in the admissibility of such a transformation, because otherwise a critical runtime error will occur and the mql4 program will be stopped.

Dynamic typecasting using dynamic_cast operator

Dynamic typecasting is performed using dynamic_cast operator that can be applied only to pointers to classes. Type validation is performed at runtime. This means that the compiler does not check the data type applied for typecasting when dynamic_cast operator is used. If a pointer is converted to a data type which is not the actual type of an object, the result is NULL.

```
dynamic_cast <type-id> ( expression )
```

The *type-id* parameter in angle brackets should point to a previously defined class type. Unlike C++, *expression* operand type can be of any value except for void.

Example:

```
class CBar { };
class CFoo : public CBar { };

void OnStart()
  {
   CBar bar;
//--- dynamic casting of *bar pointer type to *foo pointer is allowed
   CFoo *foo = dynamic_cast<CFoo *>(&bar); // no critical error
   Print(foo);                             // foo=NULL
//--- an attempt to explicitly cast a Bar type object reference to a Foo t
   foo=(CFoo *)&bar;                       // critical runtime error
   Print(foo);                             // this string is not executed
  }
```

See also

Data Types

Void Type and NULL Constant

Syntactically the void type is a fundamental type along with types of char, uchar, bool, short, ushort, int, uint, color, long, ulong, datetime, float, double and string. This type is used either to indicate that the function does not return any value, or as a function parameter it denotes the absence of parameters.

The predefined constant variable **NULL** is of the *void* type. It can be assigned to variables of any other fundamental types without conversion. The comparison of fundamental type variables with the **NULL** value is allowed.

Example:

```
//--- If the string is not initialized, then assign our predefined value t
```

Also **NULL** can be compared to pointers to objects created with the new operator.

See also

Variables, Functions

Object Pointers

In MQL4, there is a possibility to dynamically create objects of complex type. This is done by the new operator, which returns a descriptor of the created object. Descriptor is 8 bytes large. Syntactically, object descriptors in MQL4 are similar to pointers in C++.

Examples:

```
MyObject* hobject= new MyObject();
```

In contrast to C++, the hobject variable from example above is not a pointer to memory, but rather an object descriptor. Furthermore, in MQL5 all objects in function parameters must be passed by reference. Below are examples of passing objects as function parameters:

```
class Foo   {
public:
   string                m_name;
   int                   m_id;
   static int            s_counter;
   //--- constructors and desctructors
                         Foo(void){Setup("noname");};
                         Foo(string name){Setup(name);};
                        ~Foo(void){};
   //--- initializes object of type Foo
   void                  Setup(string name)
     {
      m_name=name;
      s_counter++;
      m_id=s_counter;
     }
  };
int Foo::s_counter=0;
//+-------------------------------------------------------------------+
//| Script program start function                                     |
//+-------------------------------------------------------------------+
void OnStart()
  {
//--- declare an object as variable with its automatic creation
   Foo foo1;
//--- variant of passing an object by reference
   PrintObject(foo1);

//--- declare a pointer to an object and create it using the 'new' operato
```

```
   Foo *foo2=new Foo("foo2");
//--- variant of passing a pointer to an object by reference
   PrintObject(foo2); // pointer to an object is converted automatically b

//--- declare an array of objects of type Foo
   Foo foo_objects[5];
//--- variant of passing an array of objects
   PrintObjectsArray(foo_objects); // separate function for passing an arr

//--- declare an array of pointers to objects of type Foo
   Foo *foo_pointers[5];
   for(int i=0;i<5;i++)
     {
      foo_pointers[i]=new Foo("foo_pointer");
     }
//--- variant of passing an array of pointers
   PrintPointersArray(foo_pointers); // separate function for passing an a

//--- it is obligatory to delete objects created as pointers before termin
   delete(foo2);
//--- delete array of pointers
   int size=ArraySize(foo_pointers);
   for(int i=0;i<5;i++)
      delete(foo_pointers[i]);
//---
  }
//+------------------------------------------------------------------+
//| Objects are always passed by reference                           |
//+------------------------------------------------------------------+
void PrintObject(Foo &object)
  {
   Print(__FUNCTION__,": ",object.m_id," Object name=",object.m_name);
  }
//+------------------------------------------------------------------+
//| Passing an array of objects                                      |
//+------------------------------------------------------------------+
void PrintObjectsArray(Foo &objects[])
  {
   int size=ArraySize(objects);
   for(int i=0;i<size;i++)
     {
      PrintObject(objects[i]);
     }
  }
//+------------------------------------------------------------------+
//| Passing an array of pointers to object                           |
//+------------------------------------------------------------------+
void PrintPointersArray(Foo* &objects[])
  {
   int size=ArraySize(objects);
   for(int i=0;i<size;i++)
     {
      PrintObject(objects[i]);
     }
  }
//+------------------------------------------------------------------+
```

See also

Variables, Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

References: Modifier & and Keyword this

Passing Parameters by Reference

In MQL4 parameters of simple types can be passed both by value and by reference, while parameters of compound types are always passed by reference. To inform the compiler that a parameter must be passed by reference, the ampersand character & is added before the parameter name.

Passing a parameter by reference means passing the address of the variable, that's why all changes in the parameter that is passed by reference will be immediately reflected in the source variable. Using parameter passing by reference, you can implement return of several results of a function at the same time. In order to prevent changing of a parameter passed by reference, use the const modifier.

Thus, if the input parameter of a function is an array, a structure or class object, symbol '&' is placed in the function header after the variable type and before its name.

Example

```
class CDemoClass   {
private:
   double            m_array[];

public:
   void              setArray(double &array[]);
  };
//+------------------------------------------------------------------+
//| filling the array                                                |
//+------------------------------------------------------------------+
void  CDemoClass::setArray(double &array[])
  {
   if(ArraySize(array)>0)
     {
     ArrayResize(m_array,ArraySize(array));
     ArrayCopy(m_array, array);
     }
  }
```

In the above example class CDemoClass is declared, which contains the private member - array m_array[] of double type. Function setArray() is declared, to which array[] is passed by reference. If the function header doesn't contain the indication about passing by reference, i.e. doesn't contain the ampersand character, an error message will be generated at the attempt to compile such a code.

Despite the fact that the array is passed by reference, we can't assign one array to another. We need to perform the element-wise copying of contents of the source array to the recipient array. The presence of & in the function description is the obligatory condition for arrays and structures when passed as the function parameter.

Keyword this

A variable of class type (object) can be passed both by reference and by pointer. As well as reference, the pointer allows having access to an object. After the object pointer is declared, the new operator should be applied to it to create and initialize it.

The reserved word **this** is intended for obtaining the reference of the object to itself, which is available inside class or structure methods. **this** always references to the object, in the method of which it is used, and the expression GetPointer(this) gives the pointer of the object, whose member is the function, in which call of GetPointer() is performed. In MQL4 functions can't return objects, but they can return the object pointer.

Thus, if we need a function to return an object, we can return the pointer of this object in the form of GetPointer(this). Let's add function getDemoClass() that returns pointer of the object of this class, into the description of CDemoClass.

```
class CDemoClass
  {
private:
   double             m_array[];

public:
   void               setArray(double &array[]);
   CDemoClass        *getDemoClass();
  };
//+-------------------------------------------------------------------+
//| filling the array                                                 |
//+-------------------------------------------------------------------+
void  CDemoClass::setArray(double &array[])
  {
   if(ArraySize(array)>0)
     {
      ArrayResize(m_array,ArraySize(array));
      ArrayCopy(m_array,array);
     }
  }
//+-------------------------------------------------------------------+
//| returns its own pointer                                           |
//+-------------------------------------------------------------------+
CDemoClass *CDemoClass::getDemoClass(void)
  {
   return(GetPointer(this));
  }
```

Structures don't have pointers, operators *new* and *delete* can't be applied to them, GetPointer(this) can't be used.

See also

Object Pointers, Creating and Deleting Objects, Visibility Scope and Lifetime of Variables

Operations and Expressions

Some characters and character sequences are of a special importance. These are so-called operation symbols, for example:

```
+ - * / %     Symbols of arithmetic operations && ||        Symbols of lo
= += *=       Characters assignment operators
```

Operation symbols are used in expressions and have sense when appropriate operands are given to them. Punctuation marks are emphasized, as well. These are parentheses, braces, comma, colon, and semicolon.

Operation symbols, punctuation marks, and spaces are used to separate language elements from each other.

This section contains the description of the following topics:

· Expressions

· Arithmetic Operations

· Assignment Operations

· Operations of Relation

· Boolean Operations

· Bitwise Operations

· Other Operations

· Priorities and Operations Order

Expressions

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

Examples:

```
a++; b = 10;         // several expressions are located in one line //---
x = (y * z) /
    (w + 2) + 127;
```

An expression that ends with a semicolon (;) is an operator.

See also

Precedence Rules

Arithmetic Operations

Arithmetic operations include additive and multiplicative operations:

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

Increment and decrement operations are applied only to variables, they can't be applied to constants. The prefix increment (++i) and decrement (--k) are applied to the variable right before this variable is used in an expression.

Post-increment (i++) and post-decrement (k--) are applied to the variable right after this variable is used in an expression.

Important Notice

```
int i=5;
int k = i++ + ++i;
```

Computational problems may occur while moving the above expression from one programming environment to another one (for example, from Borland C++ to MQL4). In general, the order of computations depends on the compiler implementation. In practice, there are two ways to implement the post-decrement (post-increment):

1. The post-decrement (post-increment) is applied to the variable after calculating the whole expression.
2. The post-decrement (post-increment) is applied to the variable immediately at the operation.

Currently the first way of post-decrement (post-increment) calculation is implemented in MQL4. But even knowing this peculiarity, it is not recommended to experiment with its use.

Examples:

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

See also

[Precedence Rules]

Assignment Operations

The value of the expression that includes the given operation is the value of the left operand after assignment:

```
Assigning the value of x to the y variable                               y
```

The following operations unite arithmetic or bitwise operations with operation of assignment:

```
Adding x to the y variable                                              y +
Multiplying the y variable by x                                         y *
Dividing the y variable by x                                            y /
Reminder of division of the y variable by x                             y %
Shift of the binary representation of y to the right by x bits        y >>
Shift of the binary representation of y to the left by x bits         y <<
AND bitwise operation of binary representations of y and x              y &
OR bitwise operation of binary representations of y and x               y |
Excluding OR bitwise operation of binary representations of y and x     y ^
```

Bitwise operations can be applied to integers only. When performing the operation of the logical shift of the y representation to the right/left by x bits, the 5 smallest binary digits of the x value are used, the highest ones are dropped, i.e. the shift is made to 0-31 bits.

By %= operation (y value by module of x), the result sign is equal to the sign of divided number.

The assignment operator can be used several times in an expression . In this case the processing of the expression is performed from left to right:

```
 y=x=3;
```

First, the variable x will be assigned the value 3, then the y variable will be assigned the value of x, i.e. also 3.

See also

Precedence Rules

Operations of Relation

Boolean FALSE is represented with an integer zero value, while the boolean TRUE is represented by any non-zero value.

The value of expressions containing operations of relation or logical operations is FALSE (0) or TRUE (1).

```
True if a is equal to b                      a == b; True if a is not equa
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 equal to b         a <= b;
True if a is greater than or equal to b      a >= b;
```

The equality of two real numbers can't be compared. In most cases, two seemingly identical numbers can be unequal because of different values in the 15th decimal place. In order to correctly compare two real numbers, compare the normalized difference of these numbers with zero.

Example:

```
bool CompareDoubles(double number1,double number2)
  {
   if(NormalizeDouble(number1-number2,8)==0) return(true);
   else return(false);
  }
void OnStart()
  {
   double first=0.3;
   double second=3.0;
   double third=second-2.7;
   if(first!=third)
     {
      if(CompareDoubles(first,third))
         printf("%.16f and %.16f are equal",first,third);
     }
  }
// Result: 0.3000000000000000  0.2999999999999998   are equal
```

See also

Precedence Rules

Boolean Operations

Logical Negation NOT (!)

Operand of the logical negation (!) must be of arithmetic type. The result is TRUE (1), if the operand value is FALSE (0); and it is equal to FALSE (0), if the operand differs from FALSE (0).

```
if(!a) Print("not 'a'");
```

Logical Operation OR (||)

Logical OR operation (||) of x and y values. The expression value is TRUE (1), if x or y value is true (not null). Otherwise - FALSE (0).

```
if(x<0 || x>=max_bars) Print("out of range");
```

Logical Operation AND (&&)

Logical operation AND (&&) of x and y values. The expression value is TRUE (1), if the values of x and y are true (not null). Otherwise - FALSE (0).

Brief Estimate of Boolean Operations

The scheme of the so called "brief estimate" is applied to boolean operations, i.e. the calculation of the expression is terminated when the result of the expression can be precisely estimated.

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
void OnStart()
  {
//--- the first example of the brief estimate
   if(func_false() && func_true())
     {
      Print("Operation &&: You will never see this expression");
     }
   else
     {
      Print("Operation &&: Result of the first expression is false, so the
     }
//--- the second example of the brief estimate
   if(!func_false() || !func_true())
     {
      Print("Operation ||: Result of the first expression is true, so the
     }
   else
     {
      Print("Operation ||: You will never see this expression");
     }
  }
//+------------------------------------------------------------------+
//| the function always returns false                                |
//+------------------------------------------------------------------+
bool func_false()
  {
   Print("Function func_false()");
   return(false);
  }
//+------------------------------------------------------------------+
//| the function always returns true                                 |
//+------------------------------------------------------------------+
bool func_true()
  {
   Print("Function func_true()");
   return(true);
  }
```

See also

Precedence Rules

Bitwise Operations

Complement to One

Complement of the variable value up to one. The value of the expression contains 1 in all digits where the variable value contains 0, and 0 in all digits where the variable contains 1.

```
b = ~n;
```

Example:

```
   char a='a',b;    b=~a;
   Print("a = ",a, "  b = ",b);
// The result will be:
// a = 97   b = -98
```

Right Shift

The binary representation of x is shifted to the right by y digits. If the value to shift is of the unsigned type, the logical right shift is made, i.e. the freed left-side bits will be filled with zeroes.

If the value to shift is of a sign type, the arithmetic right shift is made, i.e. the freed left-side digits will be filled with the value of a sign bit (if the number is positive, the value of the sign bit is 0; if the number is negative, the value of the sign bit is 1).

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

Example:

```
   char a='a',b='b';
   Print("Before:  a = ",a, "  b = ",b);
//--- shift to the right
   b=a>>1;
   Print("After:   a = ",a, "  b = ",b);
// The result will be:
// Before:   a = 97   b = 98
// After:    a = 97   b = 48
```

Left Shift

The binary representation of x is shifted to the left by y digits, the freed right-side digits are filled with zeros.

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

Example:

```
   char a='a',b='b';
   Print("Before:  a = ",a, "  b = ",b);
//--- shift to the left
   b=a<<1;
   Print("After:   a = ",a, "  b = ",b);
// The result will be:
// Before:  a = 97   b = 98
// After:   a = 97   b = -62
```

It is not recommended to shift by the number of bits larger or equal to the length of the variable shifted, because the result of such an operation is undefined.

Bitwise AND Operation

The bitwise AND operation of binary-coded x and y representations. The value of the expression contains a 1 (TRUE) in all digits where both x and y contain non-zero, and it contains 0 (FALSE) in all other digits.

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

Example:

```
   char a='a',b='b';
//--- AND operation
   char c=a&b;
   Print("a = ",a,"  b = ",b);
   Print("a & b = ",c);
// The result will be:
// a = 97   b = 98
// a & b = 96
```

Bitwise OR Operation

The bitwise OR operation of binary representations of x and y. The value of the expression contains 1 in all digits where x or y does not contain 0, and it contains 0 in all other digits.

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

Example:

```
   char a='a',b='b';
//--- OR operation
   char c=a|b;
   Print("a = ",a,"  b = ",b);
   Print("a | b = ",c);
// The result will be:
// a = 97   b = 98
// a | b = 99
```

Bitwise Exclusive Operation OR

The bitwise exclusive OR (eXclusive OR) operation of binary representations of x and y. The value of the expression contains a 1 in all digits where x and y have different binary values, and it contains 0 in all other digits.

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

Example:

```
   char a='a', b='b';
//--- Excluding OR operation
   char c=a^b;
   Print("a = ",a,"  b = ",b);
   Print("a ^ b = ",c);
// The result will be:
// a = 97   b = 98
// a ^ b = 3
```

Bitwise operations are performed with integers only.

See also

Precedence Rules

Other operations

Indexing ([])

When addressing the i-th element of the array, the expression value is the value of a variable with the serial number i.

Example:

```
array[i] = 3; // Assign the value of 3 to i-th element of the array.
```

Only an integer can be index of an array. Four-dimensional and below arrays are allowed. Each dimension is indexed from 0 to **dimension size**-1. In particular case, for a one-dimensional array consisting of 50 elements, the reference to the first element will look like array [0], that to the last element will be array [49].

When addressing beyond the array, the executing subsystem will generate a critical error, and the program will be stopped.

Calling Function with x1, x2 ,..., xn Arguments

Each argument can represent a constant, variable, or expression of the corresponding type. The arguments passed are separated by commas and must be inside of parentheses, the opening parenthesis must follow the name of the called function.

The expression value is the value returned by the function. If the return value is of void type, such function call cannot be placed to the right in the assignment operation. Please make sure that the expressions x1,..., xn are executed exactly in this order.

Example:

```
   int length=1000000;        string a="a",b="b",c;
//---Other Operations
   int start=GetTickCount(),stop;
   long i;
   for(i=0;i<length;i++)
     {
      c=a+b;
     }
   stop=GetTickCount();
   Print("time for 'c = a + b' = ",(stop-start)," milliseconds, i = ",i);
```

Comma Operation (,)

Expressions separated by commas are executed from left to right. All side effects of the left expression calculation can appear before the right expression is calculated. The result type and value coincide with those of the right expression. The list of parameters to be passed (see above) can be considered as an example.

Example:

```
for(i=0,j=99; i<100; i++,j--) Print(array[i][j]);
```

Dot Operator (.)

For the direct access to the public members of structures and classes the dot operation is used. Syntax:

```
Variable_name_of_structure_type.Member_name
```

Example:

```
   struct SessionTime
     {
      string sessionName;
      int    startHour;
      int    startMinutes;
      int    endHour;
      int    endMinutes;
     } st;
   st.sessionName="Asian";
   st.startHour=0;
   st.startMinutes=0;
   st.endHour=9;
   st.endMinutes=0;
```

Scope Resolution Operation (::)

Each function in a mql4 program has its own execution scope. For example, the Print() system function is performed in a global scope. Imported functions are called in the scope of the corresponding import. Method functions of classes have the scope of the corresponding class. The syntax of the scope resolution operation is as follows:

```
[Scope_name]::Function_name(parameters)
```

If there is no scope name, this is the explicit direction to use the global scope. If there is no scope resolution operation, the function is sought in the nearest scope. If there is no function in the local scope, the search is conducted in the global scope.

The scope resolution operation is also used to define function-class member.

```
type Class_name::Function_name(parameters_description)
   {
// function body
   }
```

Use of several functions of the same name from different execution contexts in a program may cause ambiguity. The priority order of function calls without explicit scope specification is the following:

1. Class methods. If no function with the specified name is set in the class, move to the next level.
2. MQL4 functions. If the language does not have such a function, move to the next level.
3. User defined global functions. If no function with the specified name is found, move to the next level.
4. Imported functions. If no function with the specified name is found, the compiler returns an error.

To avoid the ambiguity of function calls, always explicitly specify the function scope using the scope resolution operation.

Example:

```
#property script_show_inputs
#import "kernel32.dll"
   int GetLastError(void);
#import

class CCheckContext
  {
   int         m_id;
public:
               CCheckContext() { m_id=1234; }
protected:
   int         GetLastError() { return(m_id); }
  };
class CCheckContext2 : public CCheckContext
  {
   int         m_id2;
public:
               CCheckContext2() { m_id2=5678; }
   void        Print();
protected:
   int         GetLastError() { return(m_id2); }
  };
void CCheckContext2::Print()
  {
   ::Print("Terminal GetLastError",::GetLastError());
   ::Print("kernel32 GetLastError",kernel32::GetLastError());
   ::Print("parent GetLastError",CCheckContext::GetLastError());
   ::Print("our GetLastError",GetLastError());
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   CCheckContext2 test;
   test.Print();
  }
//+------------------------------------------------------------------+
```

Operation of Obtaining Data Type Size or Size of Any Data Type Object (sizeof)

Using the sizeof operation, the memory size corresponding to an identifier or type can be defined. The sizeof operation is of the following format:

Example:

```
sizeof(expression)
```

Any identifier, or type name enclosed in brackets can be used as an expression. Note that the void type name can't be used, and the identifier cannot belong to the field of bits, or be a function name.

If the expression is the name of a static array (i.e. the first dimension is given), then the result is the size of the whole array (i.e. the product of the number of elements and the length of the type). If the expression is the name of a dynamic array (the first dimension is not specified), the result will be the size of the object of the dynamic array.

When sizeof is applied to the name of structure or class type, or to the identifier of the structure or class type, the result is the actual size of the structure or class.

Example:

```
   struct myStruct
     {
      char   h;
      int    b;
      double f;
     } str;
   Print("sizeof(str) = ",sizeof(str));
   Print("sizeof(myStruct) = ",sizeof(myStruct));
```

The size is calculated at the compilation stage.

See also

Precedence Rules

Precedence Rules

Each group of operations in the table has the same priority. The higher the priority of operations is, the higher it is position of the group in the table. The precedence rules determine the grouping of operations and operands.

Attention: Precedence of operations in the MQL4 language corresponds to the priority adopted in C++.

Operation	Desciption	Execution Order
() [] .	Function Call Referencing to an array element Referencing to a structure element	From left to right
! ~ ++ -- (type) sizeof	Logical negation Bitwise negation (complement) Sign changing Increment by one Decrement by one Typecasting Determining size in bytes	Right to left
* / %	Multiplication Division Module division	From left to right
+	Addition Subtraction	From left to right
<< >>	Left shift Right shift	From left to right
< <= > >=	Less than Less than or equal Greater than Greater than or equal	From left to right
== !=	Equal Not equal	From left to right
&	Bitwise AND operation	From left to right
^	Bitwise exclusive OR	From left to right
|	Bitwise OR operation	From left to right
&&	Logical AND operation	From left to right
||	Logical OR operation	From left to right

?:	Conditional Operator	Right to left
= *= /= %= += -= <<= >>= &= ^= |=	Assignment Multiplication with assignment Division with assignment Module with assignment Addition with assignment Subtraction with assignment Left shift with assignment Right shift with assignment Bitwise AND with assignment Exclusive OR with assignment Bitwise OR with assignment	Right to left
,	Comma	From left to right

To change the operation execution order, parenthesis that are of higher priority are used.

Precedence Rules for the old version of MQL4

The precedence rules for the old version of MQL4 language are presented below.

Each group of operations in the table has the same priority. The higher is the priority, the higher is the position of the group in the table. The precedence rules determine the grouping of operations and operands.

Operation	Description	Execution Order
() []	Function call Referencing to an array element	From left to right
! - ++ -- ~	Logical negation Sign changing operation Increment Decrement Bitwise negation (complement)	From right to left
& | ^ << >>	Bitwise operation AND Bitwise operation OR Bitwise operation exclusive OR Left shift Right shift	From left to right
* / %	Multiplication Division Module division	From left to right

+ -	Addition Subtraction	From left to right
< <= > >= == !=	Less than Less than or equal Greater than Greater than or equal Equal Not equal	From left to right
||	Logical OR	From left to right
&&	Logical AND	From left to right
= += -= *= /= %= >>= <<= &= |= ^=	Assignment 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	From right to left
,	Comma	From left to right

Parentheses that have higher priority are applied to change the execution order of the operations.

Attention: Priority of performing operations in old MQL4 differs to some extent from that conventional in the C language.

Operators

Language operators describe some algorithmic operations that must be executed to accomplish a task. The program body is a sequence of such operators. Operators following one by one are separated by semicolons.

Operator	Description
Compound operator {}	One or more operators of any type, enclosed in curly braces {}
Expression operator (;)	Any expression that ends with a semicolon (;)
return operator	Terminates the current function and returns control to the calling program
if-else conditional operator	Is used when it's necessary to make a choice
?: conditional operator	A simple analog of the if-else conditional operator
switch selection operator	Passes control to the operator, which corresponds to the expression value
while loop operator	Performs an operator until the expression checked becomes false. The expression is checked before each iteration
for loop operator	Performs an operator until the expression checked becomes false. The expression is checked before each iteration
do-while loop operator	Performs an operator until the expression checked becomes false. The end condition is checked, after each loop. The loop body is always executed at least once.
break operator	Terminates the execution of the nearest attached external operator switch, while, do-while or for
continue operator	Passes control to the beginning of the nearest external loop operator while, do-while or for
new operator	Creates an object of the appropriate size and returns a descriptor of the created object.
delete operator	Deletes the object created by the new operator

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

Example:

```
if(Month() == 12)   if(Day() == 31) Print("Happy New Year!");
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Compound Operator

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

Example:

```
if(x==0)   {
   Print("invalid position x = ",x);
   return;
  }
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Expression Operator

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

Assignment Operator

Identifier = expression;

```
x=3;   y=x=3;
bool equal=(x==y);
```

Assignment operator can be used many times in an expression. In this case, the expression is processed from left to right:

Function Calling Operator

Function_name (argument1,..., argumentN);

```
FileClose(file);
```

Empty Operator

Consists only of a semicolon (;) and is used to denote an empty body of a control operator.

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Return Operator

The return operator terminates the current function execution and returns control to the calling program. The expression calculation result is returned to the calling function. The expression can contain an assignment operator.

Example:

```
int CalcSum(int x, int y)   {
   return(x+y);
  }
```

In functions with the void return type, the return operator without expression must be used:

```
void SomeFunction()
  {
   Print("Hello!");
   return;    // this operator can be removed
  }
```

The right brace of the function means implicit execution of the return operator without expression.

What can be returned: simple types, simple structures, object pointers. With the *return* operator you can't return any arrays, class objects, variables of compound structure type.

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

If-Else Conditional Operator

The IF - ELSE operator is used when a choice must be made. Formally, the syntax is as follows:

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

If the expression is true, operator1 is executed and 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 omitted else part. In this case, else addresses to the nearest previous if operator in the same block that has no else part.

Examples:

```
//--- 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>1)
  {
   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");
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Ternary Operator ?:

The general form of the ternary operator is as follows:

```
expression1 ? expression2 : expression3
```

For the first operand - "expression1" - any expression that results in a bool type value can be used. If the result is true, then the operator set by the second operand, i.e. "expression2" is executed.

If the first operand is false, the third operand - "expression3" is performed. The second and third operands, i.e. "expression2" and "expression3" should return values of one type and should not be of void type. The result of the conditional operator execution is the result of expression2 or result of the expression3, depending on the result of expression1.

```
//--- normalize difference between open and close prices for a day range d
```

This entry is equivalent to the following:

```
   double true_range;
   if(High==Low)true_range=0;               // if High and Low are equal
   else true_range=(Close-Open)/(High-Low); // if the range is not null
```

Operator Use Restrictions

Based on the value of "expression1", the operator must return one of the two values - either "expression2" or "expression3". There are several limitations to these expressions:

1. Do not mix user-defined type with simple type or enumeration. NULL can be used for the pointer.
2. If types of values are simple, the operator will be of the maximum type (see Type casting).
3. If one of the values is an enumeration and the second one is of a numeric type, the enumeration is replaced by int and the second rule is applied.
4. If both values are enumerations, their types must be identical, and the operator will be of type enumeration.

Restrictions for the user-defined types (classes or structures):

a) Types must be identical or one should be derived from the other one.

b) If types are not identical (inheritance), then the child is implicitly cast to the parent, i.e. the operator will be of the parent type.

c)Do not mix object and pointer both expressions must be either objects or pointers. NULL can be used for the pointer.

Note

Be careful when using the conditional operator as an argument of an overloaded function, because the type of the result of a conditional operator is defined at the time of program compilation. And this type is determined as the larger of the types "expression2" and "expression3".

Example:

```
void func(double d) { Print("double argument: ",d); }
void func(string s) { Print("string argument: ",s); }

bool   Expression1=true;
double Expression2=M_PI;
string Expression3="3.1415926";

void OnStart()
  {
   func(Expression2);
   func(Expression3);

   func(Expression1?Expression2:Expression3);   // warning on implicit cas
   func(!Expression1?Expression2:Expression3);  // warning on implicit cas
  }

//   Result:
//   double argument: 3.141592653589793
//   string argument: 3.1415926
//   string argument: 3.141592653589793
//   string argument: 3.1415926
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Switch Operator

Compares the expression value with constants in all the *case* variants and passes control to the operator that corresponds to the expression value. Each variant of *case* can be marked with an integer constant, a literal constant or a constant expression. The constant expression can't contain variables or function calls. Expression of the *switch* operator must be of integer type.

```
switch(expression)   {
   case constant: operators
   case constant: operators
      ...
   default: operators
  }
```

Operators marked by the *default* label are executed if none of the constants in *case* operators is equal to the expression value. The *default* variant should not be necessarily declared and should not be necessarily the last one. If none of the constants corresponds to the expression value and the *default* variant is not available, no actions are executed.

The *case* keyword with a constant are just labels, and if operators are executed for some *case* variant, the program will further execute the operators of all subsequent variants until the *break* operator occurs. It allows to bind a sequence of operators with several variants.

A constant expression is calculated during compilation. No two constants in one *switch* operator can have the same value.

Examples:

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

//---  Second example
   string res="";
   int i=0;
   switch(i)
     {
      case 1:
         res=i;break;
      default:
         res="default";break;
      case 2:
         res=i;break;
      case 3:
         res=i;break;
     }
   Print(res);
/*
   Result
   default
*/
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

While Loop Operator

The while operator consists of a checked expression and the operator, which must be fulfilled:

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

If the expression is true, the operator is executed until the expression becomes false. If the expression is false, the control is passed to the next operator. The expression value is defined before the operator is executed. Therefore, if the expression is false from the very beginning, the operator will not be executed at all.

Note

If it is expected that a large number of iterations will be handled in a loop, it is advisable that you check the fact of forced program termination using the IsStopped() function.

Example:

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

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

For Loop Operator

The for operator consists of three expressions and an executable operator:

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

Expression1 describes the loop initialization. Expression2 checks the conditions of the loop termination. If it is true, the loop body for is executed. The loop repeats expression2 until it becomes false. If it is false, the loop is terminated, and 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;
  };
```

Any of the three or all three expressions can be absent in the for operator, but the semicolons (;) that separate them must not be omitted. If expression2 is omitted, it is considered constantly true. The for(;;) operator is a continuous loop, equivalent to the while(1) operator. Each expression 1 or 3 can consist of several expressions combined by a comma operator ','.

Note

If it is expected that a large number of iterations will be handled in a loop, it is advisable that you check the fact of forced program termination using the IsStopped() function.

Examples:

```
for(x=1;x<=7000; x++)
  {
   if(IsStopped())
      break;
   Print(MathPower(x,2));
  }
//--- Another example
for(;!IsStopped();)
  {
   Print(MathPower(x,2));
   x++;
   if(x>10) break;
  }
//--- Third example
for(i=0,j=n-1;i<n && !IsStopped();i++,j--) a[i]=a[j];
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Loop Operator do while

The for and while loops check the termination at the beginning, not at the end of a loop. The third loop operator do - while checks the condition of termination at the end, after each loop iteration. The loop body is always executed at least once.

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

First the operator is executed, then the expression is calculated. If it is true, then the operator is executed again, and so on. If the expression becomes false, the loop terminates.

Note

If it is expected that a large number of iterations will be handled in a loop, it is advisable that you check the fact of forced program termination using the IsStopped() function.

Example:

```
//--- Calculate the Fibonacci series
   int counterFibonacci=15;
   int i=0,first=0,second=1;
   int currentFibonacciNumber;
   do
     {
      currentFibonacciNumber=first+second;
      Print("i = ",i,"  currentFibonacciNumber = ",currentFibonacciNumber)
      first=second;
      second=currentFibonacciNumber;
      i++; // without this operator an infinite loop will appear!
     }
   while(i<counterFibonacci && !IsStopped());
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Break Operator

The break operator terminates the execution of the nearest nested outward switch, while, do-while or for operator. The control is passed 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 for the first zero element for(i=0;i<array_size;i++)
  if(array[i]==0)
    break;
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Continue Operator

The continue operator passes control to the beginning of the nearest outward loop while, do-while or for operator, the next iteration being called. The purpose of this operator is opposite to that of break operator.

Example:

```
//--- Sum of all nonzero elements 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);
  }
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Object Create Operator new

The new operator automatically creates an object of a corresponding size, calls the object constructor and returns a descriptor of created object. In case of failure, the operator returns a null descriptor that can be compared with the NULL constant.

The new operator can be applied only to class objects. It can't be applied to structures.

The operator shall not be used to create arrays of objects. To do this, use the ArrayResize() function.

Example:

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
void CTetrisField::NewShape()
  {
   m_ypos=HORZ_BORDER;
//--- randomly create one of the 7 possible shapes
   int nshape=rand()%7;
   switch(nshape)
     {
      case 0: m_shape=new CTetrisShape1; break;
      case 1: m_shape=new CTetrisShape2; break;
      case 2: m_shape=new CTetrisShape3; break;
      case 3: m_shape=new CTetrisShape4; break;
      case 4: m_shape=new CTetrisShape5; break;
      case 5: m_shape=new CTetrisShape6; break;
      case 6: m_shape=new CTetrisShape7; break;
     }
//--- draw
   if(m_shape!=NULL)
     {
      //--- pre-settings
      m_shape.SetRightBorder(WIDTH_IN_PIXELS+VERT_BORDER);
      m_shape.SetYPos(m_ypos);
      m_shape.SetXPos(VERT_BORDER+SHAPE_SIZE*8);
      //--- draw
      m_shape.Draw();
     }
//---
  }
```

It should be noted that object descriptor is not a pointer to memory address.

An object created with the new operator must be explicitly removed using the delete operator.

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Object Delete Operator delete

The delete operator deletes an object created by the new operator, calls the corresponding class destructor and frees up memory occupied by the object. A real descriptor of an existing object is used as an operand. After the delete operation is executed, the object descriptor becomes invalid.

Example:

```
      //--- delete figure      delete m_shape;
      m_shape=NULL;
      //--- create a new figure
      NewShape();
```

See also

Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Functions

Every task can be divided into subtasks, each of which can either be directly represented in the form of a code, or divided into smaller sub-tasks. This method is called *stepwise refinement*. Functions are used for writing the code of sub-tasks to be solved. The code that describes what a function does is called *function definition:*

```
function_header   {
   instructions
  }
```

All that is before the first brace is the *header* of the function definition, and what is between braces is the *body* of the function definition. The function header includes a description of the return value type, name (identifier) and formal parameters. The number of parameters passed to the function is limited and cannot exceed 64.

The function can be called from other parts of the program as many times as necessary. In fact, the return type, function identifier and parameter types constitute the *function prototype*.

Function prototype is the function declaration, but not its definition. Due to the explicit declaration of the return type and a list of argument types, the strict type checking and implicit typecasting are possible during function calls. Very often function declarations are used in classes to improve the code readability.

The function definition must exactly match its declaration. Each declared function must be defined.

Example:

```
double                       // return value type
linfunc (double a, double b) // function name and parameter list
  {
                             // composite operator
   return (a + b);           // return value
  }
```

The return operator can return the value of an expression located in this operator. If necessary, the expression value is converted to the function result type. What can be returned: simple types, simple structures, object pointers. With the *return* operator you can't return any arrays, class objects, variables of compound structure type.

A function that returns no value should be described as that of void type.

Example:

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

Parameters passed to the function can have default values, which are defined by constants of that type.

Example:

```
int somefunc(double a,
             double d=0.0001,
             int n=5,
             bool b=true,
             string s="passed string")
  {
   Print("Required parameter a = ",a);
   Print("Pass the following parameters: d = ",d," n = ",n," b = ",b," s =
   return(0);
  }
```

If any of parameters has a default value, all subsequent parameters must also have default values.

Example of incorrect declaration:

```
int somefunc(double a,
             double d=0.0001,    // default value 0.0001 declared
             int n,              // default value is not specified !
             bool b,             // default value is not specified !
             string s="passed string")
  {
  }
```

See also

Overload, Virtual Functions, Polymorphism

Function Call

If a name that has not been described before, appears in the expression and is followed by the left parenthesis, it is contextually considered as the name of a function.

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

Arguments (formal parameters) are passed by value, i.e. 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 not guaranteed. During the execution, the system checks the number and type of arguments passed to the function. Such way of addressing to the function is called a value call.

Function call is an expression, the value of which is the value returned by the function. The function type described above must correspond with the type of the return value. The function can be declared or described in any part of the program on the global scope, i.e., outside other functions. The function cannot be declared or described inside of another function.

Examples:

```
int start()   {
   double some_array[4]={0.3, 1.4, 2.5, 3.6};
   double a=linfunc(some_array, 10.5, 8);
   //...
  }
double linfunc(double x[], double a, double b)
  {
   return (a*x[0] + b);
  }
```

At calling of a function with default parameters, the list of parameters to be passed can be limited, but not before the first default parameter.

Examples:

```
void somefunc(double init,
              double sec=0.0001, //set default values
              int level=10);
//...
somefunc();                      // Wrong call. The first parameter must b
somefunc(3.14);                  // Correct call
somefunc(3.14,0.0002);           // Correct call
somefunc(3.14,0.0002,10);        // Correct call
```

When calling a function, one may not skip parameters, even those having default values:

```
somefunc(3.14, , 10);           // Wrong call -> the second parameter was
```

Use of several functions of the same name from different execution contexts in a program may cause ambiguity. To avoid the ambiguity of function calls, always explicitly specify the function scope using scope resolution operation.

See also

Overload, Virtual Functions, Polymorphism

Passing Parameters

There are two methods, by which the machine language can pass arguments to a subprogram (function). The first method is to send a parameter by value. This method copies the argument value into a formal function parameter. Therefore, any changes in this parameter within the function have no influence on the corresponding call argument.

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
double FirstMethod(int i,int j)
  {
   double res;
//---
   i*=2;
   j/=2;
   res=i+j;
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int a=14,b=8;
   Print("a and b before call:",a," ",b);
   double d=FirstMethod(a,b);
   Print("a and b after call:",a," ",b);
  }
//--- Result of script execution
//  a and b before call: 14 8
//  a and b after call: 14 8
```

The second method is to pass by reference. In this case, reference to a parameter (not its value) is passed to a function parameter. Inside the function, it is used to refer to the actual parameter specified in the call. This means that the parameter changes will affect the argument used to call the function.

```
//+------------------------------------------------------------------+
//| Passing parameters by reference                                  |
//+------------------------------------------------------------------+
double SecondMethod(int &i,int &j)
  {
   double res;
//---
   i*=2;
   j/=2;
   res=i+j;
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int a=14,b=8;
   Print("a and b before call:",a," ",b);
   double d=SecondMethod(a,b);
   Print("a and b after call:",a," ",b);
  }
//+------------------------------------------------------------------+
//--- result of script execution
//  a and b before call: 14 8
//  a and b after call: 28 4
```

MQL4 uses both methods, with one exception: arrays, structure type variables and class objects are always passed by reference. In order to avoid changes in actual parameters (arguments passed at function call) use the access specifier const. When trying to change the contents of a variable declared with the *const* specifier, the compiler will generate an error.

Note

It should be noted that parameters are passed to a function in reversed order, i.e., first the last parameter is calculated and passed, and then the last but one, etc. The last calculated and passed parameter is the one that stands first after opening parenthesis.

Example:

```
void OnStart()
  {
//---
   int a[]={0,1,2};
   int i=0;

   func(a[i],a[i++],"First call (i = "+string(i)+")");
   func(a[i++],a[i],"Second call (i = "+string(i)+")");
// Result:
// First call (i = 0) : par1 = 1     par2 = 0
// Second call (i = 1) : par1 = 1     par2 = 1

  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void func(int par1,int par2,string comment)
  {
   Print(comment,": par1 = ",par1,"    par2 = ",par2);
  }
```

In first call (see example above) the ***i*** variable is first used in strings concatenation:

```
  "First call (i = "+string(i)+")"
```

Here its value doesn't change. Then the ***i*** variable is used in calculation of the ***a[i++]*** array element, i.e. when array element with index i is accessed, the ***i*** variable is incremented. And only after that the first parameter with changed value of ***i*** variable is calculated.

In the second call the same value of i (calculated on the first phase of function calling) is used when calculating all three parameters. Only after the first parameters is calculated the ***i*** variable is changed again.

See also

Visibility Scope and Lifetime of Variables, Overload, Virtual Functions, Polymorphism

Function Overloading

Usually the function name tends to reflect its main purpose. As a rule, readable programs contain various well selected identifiers. Sometimes different functions are used for the same purposes. Let's consider, for example, a function that calculates the average value of an array of double precision numbers and the same function, but operating with an array of integers. Both are convenient to be called AverageFromArray:

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
double AverageFromArray(const double & array[],int size)
  {
   if(size<=0) return 0.0;
   double sum=0.0;
   double aver;
//---
   for(int i=0;i<size;i++)
     {
      sum+=array[i];    // Summation for the double
     }
   aver=sum/size;       // Just divide the sum by the number
//---
   Print("Calculation of the average for an array of double type");
   return aver;
  }
//+------------------------------------------------------------------+
//| The calculation of average for an array of int type              |
//+------------------------------------------------------------------+
double AverageFromArray(const int & array[],int size)
  {
   if(size<=0) return 0.0;
   double aver=0.0;
   int sum=0;
//---
   for(int i=0;i<size;i++)
     {
      sum+=array[i];     // Summation for the int
     }
   aver=(double)sum/size;// Give the amount of type double, and divide
//---
   Print("Calculation of the average for an array of int type");
   return aver;
  }
```

Each function contains the message output via the Print() function;

```
   Print("Calculation of the average for an array of int type");
```

The compiler selects a necessary function in accordance with the types of arguments and their quantity. The rule, according to which the choice is made, is called the *signature matching algorithm*. A signature is a list of types used in the function declaration.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int    a[5]={1,2,3,4,5};
   double b[5]={1.1,2.2,3.3,4.4,5.5};
   double int_aver=AverageFromArray(a,5);
   double double_aver=AverageFromArray(b,5);
   Print("int_aver = ",int_aver,"   double_aver = ",double_aver);
  }
//--- Result of the script
// Calculate the average for an array of int type
// Calculate the average for an array of double type
// int_aver= 3.00000000    double_aver= 3.30000000
```

Function overloading is a process of creating several functions with the same name, but different parameters. This means that in overloaded variants of a function, the number of arguments and/or their type must be different. A specific function variant is selected based on the correspondence of the list of arguments when calling the function, to the list of parameters in the function declaration.

When an overloaded function is called, the compiler must have an algorithm to select the appropriate function. The algorithm that performs this choice depends on castings of what types are present. The best correspondence must be unique. An overloaded function must be the best match among all the other variants for at least one argument. At the same time it must match for all other arguments not worse than other variants.

Below is a matching algorithm for each argument.

Algorithm of Choosing an Overloaded Function

1.Use strict matching (if possible).

2.Try standard type increase.

3.Try standard typecasting.

The standard type increase is better than other standard conversions. Increase is the conversion of float to double, of bool, char, short or enum to int. Typecasting of arrays of similar integer types also belongs to typecasting. Similar types are: bool, char, uchar, since all the three types are single-byte integers; double-byte integers short and ushort; 4-byte integers int, uint, and color; long, ulong, and datetime.

Of course, the strict matching is the best. To achieve such a consistency typecasting can be used. The compiler cannot cope with ambiguous situations. Therefore you should not rely on subtle differences of types and implicit conversions that make the overloaded function unclear.

If you doubt, use explicit conversion to ensure strict compliance.

Examples of overloaded functions in MQL4 can be seen in the example of ArrayInitialize() functions.

Function overloading rules apply to overload of class methods.

Overloading of system functions is allowed, but it should be observed that the compiler is able to accurately select the necessary function. For example, we can overload the system function MathMax() in 4 different ways, but only two variants are correct.

Example:

```
// 1. overload is allowed - function differs from built-in MathMax() funct
double MathMax(double a,double b,double c);

// 2. overload is not allowed!
// number of parameters is different, but the last has a default value
// this leads to the concealment of the system function when calling, whic
double MathMax(double a,double b,double c=DBL_MIN);

// 3. overload is allowed - normal overload by type of parameters a and b
double MathMax(int a,int b);

// 4. overload is not allowed!
// the number and types of parameters are the same as in original double M
int MathMax(double a,double b);
```

See also

Overload, Virtual Functions, Polymorphism

Operation Overloading

For ease of code reading and writing, overloading of some operations is allowed. Overloading operator is written using the keyword operator. The following operators can be overloaded:

- binary +,-,/,*,%,<<,>>,==,!=,<,>,<=,>=,=,+=,-=,/=,*=,%=,&=,|=,^=, <<=,>>=,&&,||,&,|,^
- unary +,-,++,--,!,~
- assignment operator =
- indexing operator []

Operation overloading allows the use of the operating notation (written in the form of simple expressions) for complex objects - structures and classes. Writing expressions using overloaded operations simplifies the view of the source code, because a more complex implementation is hidden.

For example, consider complex numbers, which consist of real and imaginary parts. They are widely used in mathematics. The MQL4 language has no data type to represent complex numbers, but it is possible to create a new data type in the form of a structure or class. Declare the complex structure and define four methods that implement four arithmetic operations:

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
struct complex
  {
   double               re; // Real part
   double               im; // Imaginary part
   //--- Constructors
                        complex():re(0.0),im(0.0) {   }
                        complex(const double r):re(r),im(0.0) {   }
                        complex(const double r,const double i):re(r),im(i) {
                        complex(const complex &o):re(o.re),im(o.im) { }
   //--- Arithmetic operations
   complex              Add(const complex &l,const complex &r) const;   // Add
   complex              Sub(const complex &l,const complex &r) const;   // Sub
   complex              Mul(const complex &l,const complex &r) const;   // Mul
   complex              Div(const complex &l,const complex &r) const;   // Div
  };
```

Now, in our code we can declare variables representing complex numbers, and work with them.

For example:

```
void OnStart()
  {
//--- Declare and initialize variables of a complex type
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
//--- Sum up two numbers
   complex z;
   z=a.Add(a,b);
   PrintFormat("a+b=%.2f+i*%.2f",z.re,z.im);
//--- Multiply two numbers
   z=a.Mul(a,b);
   PrintFormat("a*b=%.2f+i*%.2f",z.re,z.im);
//--- Divide two numbers
   z=a.Div(a,b);
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
  }
```

But it would be more convenient to use usual operators "+", "-", "*" and "/" for ordinary arithmetic operations with complex numbers.

Keyword operator is used for defining a member function that performs type conversion. Unary and binary operations for class object variables can be overloaded as non-static member functions. They implicitly act on the class object.

Most binary operations can be overloaded like regular functions that take one or both arguments as a class variable or a pointer to an object of this class. For our type complex, overloading in the declaration will look like this:

```
   //--- Operators
   complex operator+(const complex &r) const { return(Add(this,r)); }
   complex operator-(const complex &r) const { return(Sub(this,r)); }
   complex operator*(const complex &r) const { return(Mul(this,r)); }
   complex operator/(const complex &r) const { return(Div(this,r)); }
```

The full example of the script:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Declare and initialize variables of type complex
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
   //a.re=5;
   //a.im=1;
   //b.re=-1;
   //b.im=-5;
//--- Sum up two numbers
   complex z=a+b;
   PrintFormat("a+b=%.2f+i*%.2f",z.re,z.im);
//--- Multiply two numbers

   z=a*b;
   PrintFormat("a*b=%.2f+i*%.2f",z.re,z.im);
//--- Divide two numbers
   z=a/b;
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
  }
//+------------------------------------------------------------------+
//| A structure for operations with complex numbers                  |
//+------------------------------------------------------------------+
struct complex
  {
   double            re; // Real part
   double            im; // Imaginary part
   //--- Constructors
                     complex():re(0.0),im(0.0) {  }
                     complex(const double r):re(r),im(0.0) {  }
                     complex(const double r,const double i):re(r),im(i) {
                     complex(const complex &o):re(o.re),im(o.im) { }
   //--- Arithmetic operations
   complex           Add(const complex &l,const complex &r) const;  // Add
   complex           Sub(const complex &l,const complex &r) const;  // Sub
   complex           Mul(const complex &l,const complex &r) const;  // Mul
   complex           Div(const complex &l,const complex &r) const;  // Div
   //--- Binary operators
   complex operator+(const complex &r) const { return(Add(this,r)); }
   complex operator-(const complex &r) const { return(Sub(this,r)); }
   complex operator*(const complex &r) const { return(Mul(this,r)); }
   complex operator/(const complex &r) const { return(Div(this,r)); }
  };
//+------------------------------------------------------------------+
//| Addition                                                         |
//+------------------------------------------------------------------+
complex complex::Add(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re+r.re;
   res.im=l.im+r.im;
//--- Result
   return res;
  }
//+------------------------------------------------------------------+
//| Subtraction                                                      |
//+------------------------------------------------------------------+
complex complex::Sub(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re-r.re;
   res.im=l.im-r.im;
//--- Result
   return res;
  }
//+------------------------------------------------------------------+
//| Multiplication                                                   |
//+------------------------------------------------------------------+
complex complex::Mul(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re*r.re-l.im*r.im;
   res.im=l.re*r.im+l.im*r.re;
//--- Result
   return res;
  }
//+------------------------------------------------------------------+
//| Division                                                         |
//+------------------------------------------------------------------+
complex complex::Div(const complex &l,const complex &r) const
  {
//--- Empty complex number
   complex res(EMPTY_VALUE,EMPTY_VALUE);
//--- Check for zero
   if(r.re==0 && r.im==0)
     {
      Print(__FUNCTION__+": number is zero");
      return(res);
     }
//--- Auxiliary variables
   double e;
   double f;
//--- Selecting calculation variant
   if(MathAbs(r.im)<MathAbs(r.re))
     {
      e = r.im/r.re;
      f = r.re+r.im*e;
      res.re=(l.re+l.im*e)/f;
      res.im=(l.im-l.re*e)/f;
     }
   else
     {
      e = r.re/r.im;
      f = r.im+r.re*e;
      res.re=(l.im+l.re*e)/f;
      res.im=(-l.re+l.im*e)/f;
     }
//--- Result
   return res;
  }
```

Most unary operations for classes can be overloaded as ordinary functions that accept a single class object argument or a pointer to it. Add overloading of unary operations "-" and "!".

```
//+------------------------------------------------------------------+
//| Structure for operations with complex numbers                    |
//+------------------------------------------------------------------+
struct complex
  {
   double            re;       // Real part
   double            im;       // Imaginary part
...
   //--- Unary operators
   complex operator-()  const; // Unary minus
   bool    operator!()  const; // Negation
  };
...
//+------------------------------------------------------------------+
//| Overloading the "unary minus" operator                           |
//+------------------------------------------------------------------+
complex complex::operator-() const
  {
   complex res;
//---
   res.re=-re;
   res.im=-im;
//--- Result
   return res;
  }
//+------------------------------------------------------------------+
//| Overloading the "logical negation" operator                      |
//+------------------------------------------------------------------+
bool complex::operator!() const
  {
//--- Are the real and imaginary parts of the complex number equal to zero
   return (re!=0 && im!=0);
  }
```

Now we can check the value of a complex number for zero and get a negative value:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Declare and initialize variables of type complex
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
//--- Divide the two numbers
   complex z=a/b;
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//--- A complex number is equal to zero by default (in the default constru
   complex zero;
   Print("!zero=",!zero);
//--- Assign a negative value
   zero=-z;
   PrintFormat("z=%.2f+i*%.2f,  zero=%.2f+i*%.2f",z.re,z.im, zero.re,zero.
   PrintFormat("-zero=%.2f+i*%.2f",-zero.re,-zero.im);
//--- Check for zero once again
   Print("!zero=",!zero);
//---
  }
```

Note that we did not have to overload the assignment operator "=", as structures of simple types can be directly copied one into each other. Thus, we can now write a code for calculations involving complex numbers in the usual manner.

Overloading of the indexing operator allows to obtain the values of the arrays enclosed in an object, in a simple and familiar way, and it also contributes to a better readability of the source code. For example, we need to provide access to a symbol in the string at the specified position. A string in MQL4 is a separate type string, which is not an array of symbols, but with the help of an overloaded indexing operation we can provide a simple and transparent work in the generated CString class:

```
//+------------------------------------------------------------------+
//| Class to access symbols in string as in array of symbols         |
//+------------------------------------------------------------------+
class CString
  {
   string            m_string;

public:
                     CString(string str=NULL):m_string(str) { }
   ushort operator[] (int x) { return(StringGetCharacter(m_string,x)); }
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- An array for receiving symbols from a string
   int     x[]={ 19,4,18,19,27,14,15,4,17,0,19,14,17,27,26,28,27,5,14,
                 17,27,2,11,0,18,18,27,29,30,19,17,8,13,6 };
   CString str("abcdefghijklmnopqrstuvwxyz[ ]CS");
   string  res;
//--- Make up a phrase using symbols from the str variable
   for(int i=0,n=ArraySize(x);i<n;i++)
     {
      res+=ShortToString(str[x[i]]);
     }
//--- Show the result
   Print(res);
  }
```

Another example of overloading of the indexing operation is operations with matrices. The matrix represents a two-dimensional dynamic array, the array size is not defined in advance. Therefore, you cannot declare an array of form array[][] without specifying the size of the second dimension, and then pass this array as a parameter. A possible solution is a special class CMatrix, which contains an array of CRow class objects.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Operations of addition and multiplication of matrices
   CMatrix A(3),B(3),C();
//--- Prepare an array for rows
```

```
   double a1[3]={1,2,3}, a2[3]={2,3,1}, a3[3]={3,1,2};
   double b1[3]={3,2,1}, b2[3]={1,3,2}, b3[3]={2,1,3};
//--- Fill the matrices
   A[0]=a1; A[1]=a2; A[2]=a3;
   B[0]=b1; B[1]=b2; B[2]=b3;
//--- Output the matrices in the Experts log
   Print("---- Elements of matrix A");
   Print(A.String());
   Print("---- Elements of matrix B");
   Print(B.String());

//--- Addition of matrices
   Print("---- Addition of matrices A and B");
   C=A+B;
//--- Output the formatted string representation
   Print(C.String());

//--- Multiplication of matrices
   Print("---- Multiplication of matrices A and B");
   C=A*B;
   Print(C.String());

//--- Now we show how to get values in the style of dynamic arrays matrix[
   Print("Output the values of matrix C elementwise");
//--- Go through the matrix rows - CRow objects - in a loop
   for(int i=0;i<3;i++)
     {
      string com="| ";
      //--- Form rows from the matrix for the value
      for(int j=0;j<3;j++)
        {
         //--- Get the matrix element by the number of the row and column
         double element=C[i][j];// [i] - Access to CRow in the array m_row
                                // [j] - Overloaded operator of indexing i
         com=com+StringFormat("a(%d,%d)=%G ; ",i,j,element);
        }
      com+="|";
      //--- Output the values of the row
      Print(com);
     }
  }
//+------------------------------------------------------------------+
//| Class "Row"                                                      |
//+------------------------------------------------------------------+
class CRow
  {
private:
   double             m_array[];
public:
   //--- Constructors and a destructor
                     CRow(void)          { ArrayResize(m_array,0);    }
                     CRow(const CRow &r) { this=r;                    }
                     CRow(const double &array[]);
                    ~CRow(void){};
   //--- Number of elements in the row
   int               Size(void) const    { return(ArraySize(m_array));}
   //--- Returns a string with values
   string            String(void) const;
   //--- Indexing operator
   double            operator[](int i) const  { return(m_array[i]);   }
   //--- Assignment operators
   void              operator=(const double  &array[]); // An array
   void              operator=(const CRow & r);         // Another CRow ob
   double            operator*(const CRow &o);          // CRow object for
  };
//+------------------------------------------------------------------+
//| Constructor for initializing a row with an array                 |
//+------------------------------------------------------------------+
void  CRow::CRow(const double &array[])
  {
   int size=ArraySize(array);
//--- If the array is not empty
   if(size>0)
     {
      ArrayResize(m_array,size);
      //--- Fill with values
      for(int i=0;i<size;i++)
         m_array[i]=array[i];
     }
//---
  }
//+------------------------------------------------------------------+
//| Assignment operation for the array                               |
//+------------------------------------------------------------------+
void CRow::operator=(const double &array[])
  {
   int size=ArraySize(array);
   if(size==0) return;
//--- Fill the array with values
   ArrayResize(m_array,size);
   for(int i=0;i<size;i++) m_array[i]=array[i];
//---
  }
//+------------------------------------------------------------------+
//| Assignment operation for CRow                                    |
//+------------------------------------------------------------------+
void CRow::operator=(const CRow  &r)
  {
   int size=r.Size();
   if(size==0) return;
//--- Fill the array with values
   ArrayResize(m_array,size);
   for(int i=0;i<size;i++) m_array[i]=r[i];
//---
  }
//+------------------------------------------------------------------+
//| Operator of multiplication by another row                        |
//+------------------------------------------------------------------+
double CRow::operator*(const CRow &o)
  {
   double res=0;
//--- Verifications
   int size=Size();
   if(size!=o.Size() || size==0)
     {
      Print(__FUNCSIG__,": Failed to multiply two matrices, their sizes ar
      return(res);
     }
//--- Multiply arrays elementwise and add the products
   for(int i=0;i<size;i++)
      res+=m_array[i]*o[i];
//--- Result
   return(res);
  }
//+------------------------------------------------------------------+
//| Returns a formatted string representation                        |
//+------------------------------------------------------------------+
string CRow::String(void) const
  {
   string out="";
//--- If the size of the array is greater than zero
   int size=ArraySize(m_array);
//--- We work only with a non-zero number of array elements
   if(size>0)
     {
      out="{";
      for(int i=0;i<size;i++)
        {
         //--- Collect the values to a string
         out+=StringFormat(" %G;",m_array[i]);
        }
      out+=" }";
     }
//--- Result
   return(out);
  }
//+------------------------------------------------------------------+
//| Class "Matrix"                                                   |
//+------------------------------------------------------------------+
class CMatrix
  {
private:
   CRow              m_rows[];

public:
   //--- Constructors and a destructor
                     CMatrix(void);
                     CMatrix(int rows)  { ArrayResize(m_rows,rows);
                    ~CMatrix(void){};
   //--- Get the matrix sizes
   int               Rows()       const { return(ArraySize(m_rows));
   int               Cols()       const { return(Rows()>0? m_rows[0].Size(
   //--- Returns the value of the column in the form of a CRow row
   CRow              GetColumnAsRow(const int col_index) const;
   //--- Returns a string with the matrix values
   string            String(void) const;
   //--- The indexing operator returns a string by its number
   CRow *operator[](int i) const        { return(GetPointer(m_rows[i]));
   //--- Addition operator
   CMatrix           operator+(const CMatrix &m);
   //--- Multiplication operator
   CMatrix           operator*(const CMatrix &m);
   //--- Assignment operator
   CMatrix          *operator=(const CMatrix &m);
  };
//+------------------------------------------------------------------+
//| Default constructor, create and array of rows of zero size       |
//+------------------------------------------------------------------+
CMatrix::CMatrix(void)
  {
//--- The zero number of rows in the matrix
   ArrayResize(m_rows,0);
//---
  }
//+------------------------------------------------------------------+
//| Returns the column value in the form of CRow                     |
//+------------------------------------------------------------------+
CRow  CMatrix::GetColumnAsRow(const int col_index) const
```

```
  {
//--- A variable to get the values from the column
   CRow row();
//--- The number of rows in the matrix
   int rows=Rows();
//--- If the number of rows is greater than zero, execute the operation
   if(rows>0)
     {
      //--- An array to receive the values of the column with index col_in
      double array[];
      ArrayResize(array,rows);
      //--- Filling the array
      for(int i=0;i<rows;i++)
        {
         //--- Check the number of the column for row i - it may exceed th
         if(col_index>=this[i].Size())
           {
            Print(__FUNCSIG__,": Error! Column number ",col_index,"> row s
            break; // row will be uninitialized object
           }
         array[i]=this[i][col_index];
        }
      //--- Create a CRow row based on the array values
      row=array;
     }
//--- Result
   return(row);
  }
//+------------------------------------------------------------------+
//| Addition of two matrices                                         |
//+------------------------------------------------------------------+
CMatrix CMatrix::operator+(const CMatrix &m)
  {
//--- The number of rows and columns in the passed matrix
   int cols=m.Cols();
   int rows=m.Rows();
//--- The matrix to receive the addition results
   CMatrix res(rows);
//--- The sizes of the matrix must match
   if(cols!=Cols() || rows!=Rows())
     {
      //--- Addition impossible
      Print(__FUNCSIG__,": Failed to add two matrices, their sizes are dif
      return(res);
     }
//--- Auxiliary array
   double arr[];
   ArrayResize(arr,cols);
//--- Go through rows to add
   for(int i=0;i<rows;i++)
     {
      //--- Write the results of addition of matrix strings in the array
      for(int k=0;k<cols;k++)
        {
         arr[k]=this[i][k]+m[i][k];
        }
      //--- Place the array to the matrix row
      res[i]=arr;
     }
//--- return the result of addition of matrices
   return(res);
  }
//+------------------------------------------------------------------+
//| Multiplication of two matrices                                   |
//+------------------------------------------------------------------+
CMatrix CMatrix::operator*(const CMatrix &m)
  {
//--- Number of columns of the first matrix, number of rows passed in the
   int cols1=Cols();
   int rows2=m.Rows();
   int rows1=Rows();
   int cols2=m.Cols();
//--- Matrix to receive the addition result
   CMatrix res(rows1);
//--- Matrices should be coordinated
   if(cols1!=rows2)
     {
      //--- Multiplication impossible
      Print(__FUNCSIG__,": Failed to multiply two matrices, format is not
            "- number of columns in the first factor should be equal to th
      return(res);
     }
//--- Auxiliary array
   double arr[];
   ArrayResize(arr,cols1);
//--- Fill the rows in the multiplication matrix
   for(int i=0;i<rows1;i++)// Go through rows
     {
      //--- Reset the receiving array
      ArrayInitialize(arr,0);
      //--- Go through elements in the row
      for(int k=0;k<cols1;k++)
        {
         //--- Take values of column k of the matrix m in the for of CRow
         CRow column=m.GetColumnAsRow(k);
         //--- Multiply two rows and write the result of scalar multiplica
         arr[k]=this[i]*column;
        }
      //--- place array arr[] in the i-th row of the matrix
      res[i]=arr;
     }
//--- Return the product of two matrices
   return(res);
  }
//+------------------------------------------------------------------+
//| Assignment operation                                             |
//+------------------------------------------------------------------+
CMatrix *CMatrix::operator=(const CMatrix &m)
  {
//--- Find and set the number of rows
   int rows=m.Rows();
   ArrayResize(m_rows,rows);
//--- Fill our rows with the values of rows of  the passed matrix
   for(int i=0;i<rows;i++) this[i]=m[i];
//---
   return(GetPointer(this));
  }
//+------------------------------------------------------------------+
//| String representation of the matrix                              |
//+------------------------------------------------------------------+
string CMatrix::String(void) const
  {
   string out="";
   int rows=Rows();
//--- Form string by string
   for(int i=0;i<rows;i++)
     {
      out=out+this[i].String()+"\r\n";
     }
//--- Result
   return(out);
  }
```

See also

Overloading, Arithmetic Operations, Function Overloading, Precedence Rules

Description of External Functions

External functions defined in another module must be explicitly described. The description includes returned type, function name and series of input parameters with their types. The absence of such a description can lead to errors when compiling, building, or executing a program. When describing an external object, use the keyword *#import* indicating the module.

Examples:

```
#import "user32.dll"   int     MessageBoxW(int hWnd ,string szText,string
  int      SendMessageW(int hWnd,int Msg,int wParam,int lParam);
#import "lib.ex4"
  double   round(double value);
#import
```

With the help of import, it is easy to describe functions that are called from external DLL or compiled EX4 libraries. EX4 libraries are compiled ex4 files, which have the library property. Only function described with the export modifier can be imported from EX4 libraries.

Please keep in mind that DLL and EX4 libraries should have different names (regardless of the directories they are located in) if they are imported together. All imported functions have the scope resolution corresponding to the library's "file name".

Use of several functions of the same name from different execution contexts in a program may cause ambiguity. To avoid the ambiguity of function calls, always explicitly specify the function scope using the scope resolution operation.

Example:

```
#import "kernel32.dll"
   int GetLastError();
#import "lib.ex4"
   int GetLastError();
#import

class CFoo
  {
public:
   int            GetLastError() { return(12345); }
   void           func()
     {
      Print(GetLastError());           // call of the class method
      Print(::GetLastError());         // call of the MQL5 function
      Print(kernel32::GetLastError()); // call of the DLL library function
      Print(lib::GetLastError());      // call of the EX4 library function
     }
  };

void OnStart()
  {
   CFoo foo;
   foo.func();
  }
```

See also

Overload, Virtual Functions, Polymorphism

Exporting Functions

A function declared in a mql4 program with the *export* postmodifier can be used in another mql4 program. Such a function is called exportable, and it can be called from other programs after compilation.

```
int Function() export   {
  }
```

This modifier orders the compiler to add the function into the table of EX4 functions exported by this ex4 file. Only function with such a modifier are accessible ("visible") from other mql4 programs.

The library property tells the compiler that the EX4-file will be a library, and the compiler will show it in the header of EX4.

All functions that are planned as exportable ones must be marked with the *export* modifier.

When compiling libraries in the strict mode, the export modifier should be added for each exported function, otherwise the function will not be accessible from outside.

See also

Overload, Virtual Functions, Polymorphism

Event Handling Functions

The MQL4 language provides processing of some predefined events. Functions for handling these events must be defined in a MQL4 program; function name, return type, composition of parameters (if there are any) and their types must strictly conform to the description of the event handler function.

The event handler of the client terminal identifies functions, handling this or that event, by the type of return value and type of parameters. If other parameters, not corresponding to below descriptions, are specified for a corresponding function, or another return type is indicated for it, such a function will not be used as an event handler.

OnStart

The OnStart() function is the Start event handler, which is automatically generated **only** for running **scripts**. It must be of void type, with no parameters:

```
void OnStart();
```

For the OnStart() function, the int return type can be specified.

OnInit

The OnInit() function is the Init event handler. It must be of void or int type, with no parameters:

```
void OnInit();
```

The Init event is generated immediately after an Expert Advisor or an indicator is downloaded; The OnInit() function is used for initialization. If OnInit() has the int type of the return value, the non-zero return code means unsuccessful initialization, and it generates the Deinit event with the code of deinitialization reason REASON_INITFAILED.

OnInit() function execution result is analyzed by the terminal's runtime subsystem only if the program has been compiled using #property strict.

To optimize input parameters of an Expert Advisor, it is recommended to use values of the ENUM_INIT_RETCODE enumeration as the return code.. During initialization of an Expert Advisor before the start of testing you can request information about the configuration and resources using the TerminalInfoInteger() function.

ENUM_INIT_RETCODE

Identifier	Description
INIT_SUCCEEDED	Successful initialization, testing of the Expert Advisor can be continued. This code means the same as a null value the Expert Advisor has been successfully initialized in the tester.
INIT_FAILED	Initialization failed; there is no point in continuing testing because of fatal errors. For example, failed to create an indicator that is required for the work of the Expert Advisor. This return value means the same as a value other than zero - initialization of the Expert Advisor in the tester failed.
INIT_PARAMETERS_INCORRECT	This value means the incorrect set of input parameters. The result string containing this return code is highlighted in red in the general optimization table. Testing for the given set of parameters of the Expert Advisor will not be executed

The OnInit() function of the void type always denotes successful initialization.

OnDeinit

The OnDeinit() function is called during deinitialization and is the Deinit event handler. It must be declared as the void type and should have one parameter of the const int type, which contains the code of deinitialization reason. If a different type is declared, the compiler will generate a warning, but the function will not be called.

```
void OnDeinit(const int reason);
```

The Deinit event is generated for Expert Advisors and indicators in the following cases:

- before reinitialization due to the change of a symbol or chart period, to which the mql4 program is attached;
- before reinitialization due to the change of input parameters;
- before unloading the mql4 program.

OnTick

The NewTick event is generated for **Expert Advisors only** when a new tick for a symbol is received, to the chart of which the Expert Advisor is attached. It's useless to define the OnTick() function in a custom indicator or script, because the NewTick event is not generated for them.

The Tick event is generated only for Expert Advisors, but this does not mean that Expert Advisors required the OnTick() function, since not only NewTick events are generated for Expert Advisors, but also events of Timer, BookEvent and ChartEvent are generated. It must be declared as the void type, with no parameters:

```
void OnTick();
```

OnTimer

The OnTimer() function is called when the Timer event occurs, which is generated by the system timer only for Expert Advisors and indicators - it can't be used in scripts. The frequency of the event occurrence is set when subscribing to notifications about this event to be received by the EventSetTimer() function.

You can unsubscribe from receiving timer events for a particular Expert Advisor using the EventKillTimer() function. The function must be defined with the void type, with no parameters:

```
void OnTimer();
```

It is recommended to call the EventSetTimer() function once in the OnInit() function, and the EventKillTimer() function should be called once in OnDeinit().

Every Expert Advisor, as well as every indicator works with its own timer and receives events only from it. As soon as the mql4 program stops operating, the timer is destroyed forcibly, if it was created but hasn't been disabled by the EventKillTimer() function.

OnTester

The OnTester() function is the handler of the Tester event that is automatically generated after a history testing of an Expert Advisor on the chosen interval is over. The function must be defined with the double type, with no parameters:

```
double OnTester();
```

The function is called right before the call of OnDeinit() and has the same type of the return value - double. OnTester() can be used only in the testing of Expert Advisors. Its main purpose is to calculate a certain value that is used as the Custom max criterion in the genetic optimization of input parameters.

In the genetic optimization descending sorting is applied to results within one generation. I.e. from the point of view of the optimization criterion, the best results are those with largest values (for the Custom max optimization criterion values returned by the OnTester function are taken into account). In such a sorting, the worst values are positioned at the end and further thrown off and do not participate in the forming of the next generation.

OnChartEvent

OnChartEvent() is the handler of a group of ChartEvent events:

- CHARTEVENT_KEYDOWN event of a keystroke, when the chart window is focused;
- CHARTEVENT_MOUSE_MOVE mouse move events and mouse click events (if CHART_EVENT_MOUSE_MOVE=true is set for the chart);
- CHARTEVENT_OBJECT_CREATE event of graphical object creation (if CHART_EVENT_OBJECT_CREATE=true is set for the chart);
- CHARTEVENT_OBJECT_CHANGE event of change of an object property via the properties dialog;
- CHARTEVENT_OBJECT_DELETE event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE=true is set for the chart);
- CHARTEVENT_CLICK event of a mouse click on the chart;
- CHARTEVENT_OBJECT_CLICK event of a mouse click in a graphical object belonging to the chart;
- CHARTEVENT_OBJECT_DRAG event of a graphical object move using the mouse;
- CHARTEVENT_OBJECT_ENDEDIT event of the finished text editing in the entry box of the LabelEdit graphical object;
- CHARTEVENT_CHART_CHANGE event of chart changes;
- CHARTEVENT_CUSTOM+n ID of the user event, where n is in the range from 0 to 65535.
- CHARTEVENT_CUSTOM_LAST the last acceptable ID of a custom event (CHARTEVENT_CUSTOM +65535).

The function can be called only in Expert Advisors and indicators. The function should be of void type with 4 parameters:

```
void OnChartEvent(const int id,          // Event ID                   cons
                  const double& dparam, // Parameter of type double event
                  const string& sparam  // Parameter of type string events
  );
```

For each type of event, the input parameters of the OnChartEvent() function have definite values that are required for the processing of this event. The events and values passed through these parameters are listed in the table below.

Event	Value of the id parameter	Value of the lpar parameter
Event of a keystroke	CHARTEVENT_KEYDOWN	code of a presse key
Mouse events (if property CHART_EVENT_MOUSE_MOVE=true is set for the chart)	CHARTEVENT_MOUSE_MOVE	the X coordinate
Event of graphical object creation (if CHART_EVENT_OBJECT_CREATE=true is set for the chart)	CHARTEVENT_OBJECT_CREATE	
Event of change of an object property via the properties dialog	CHARTEVENT_OBJECT_CHANGE	
Event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)	CHARTEVENT_OBJECT_DELETE	
Event of a mouse click on the chart	CHARTEVENT_CLICK	the X coordinate
Event of a mouse click in a graphical object belonging to the chart	CHARTEVENT_OBJECT_CLICK	the X coordinate
Event of a graphical object dragging using the mouse	CHARTEVENT_OBJECT_DRAG	
Event of the finished text editing in the entry box of the LabelEdit graphical object	CHARTEVENT_OBJECT_ENDEDIT	
Event of chart Changes	CHARTEVENT_CHART_CHANGE	

ID of the user event under the N number	CHARTEVENT_CUSTOM+N	Value set by the EventChartCusto function

OnCalculate

The OnCalculate() function is called only in custom indicators when it's necessary to calculate the indicator values by the Calculate event. This usually happens when a new tick is received for the symbol, for which the indicator is calculated. This indicator is not required to be attached to any price chart of this symbol.

The OnCalculate() function must have a return type int.

```
int OnCalculate (const int rates_total,      // size of input time series
                 const int prev_calculated,  // bars handled in previous c
                 const datetime& time[],     // Time
                 const double& open[],       // Open
                 const double& high[],       // High
                 const double& low[],        // Low
                 const double& close[],      // Close
                 const long& tick_volume[],  // Tick Volume
                 const long& volume[],       // Real Volume
                 const int& spread[]         // Spread
   );
```

Parameters of open[], high[], low[] and close[] contain arrays with open prices, high and low prices and close prices of the current time frame. The time[] parameter contains an array with open time values, the spread[] parameter has an array containing the history of spreads (if any spread is provided for the traded security). The parameters of volume[] and tick_volume[] contain the history of trade and tick volume, respectively.

To determine the indexing direction of time[], open[], high[], low[], close[], tick_volume[], volume[] and spread[], call ArrayGetAsSeries(). In order not to depend on default values, you should unconditionally call the ArraySetAsSeries() function for those arrays, which are expected to work with.

The first rates_total parameter contains the number of bars, available to the indicator for calculation, and corresponds to the number of bars available in the chart.

We should note the connection between the return value of OnCalculate() and the second input parameter prev_calculated. During the function call, the prev_calculated parameter contains a value **returned** by OnCalculate() during **previous** call. This allows for economical algorithms for calculating the custom indicator in order to avoid repeated calculations for those bars that haven't changed since the previous run of this function.

For this, it is usually enough to return the value of the rates_total parameter, which contains the number of bars in the current function call. If since the last call of OnCalculate() price data has changed (a deeper history downloaded or history blanks filled), the value of the input parameter prev_calculated will be set to zero by the terminal.

To understand it better, it would be useful to start the indicator, which code is attached below.

Indicator Example:

```
#property indicator_chart_window
#property indicator_buffers 1
//---- plot Line
#property indicator_label1  "Line"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDarkBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- indicator buffers
double         LineBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime& time[],
                const double& open[],
                const double& high[],
                const double& low[],
                const double& close[],
                const long& tick_volume[],
                const long& volume[],
                const int& spread[])
  {
//--- Get the number of bars available for the current symbol and chart pe
   int bars=Bars(Symbol(),0);
   Print("Bars = ",bars,", rates_total = ",rates_total,",  prev_calculated
   Print("time[0] = ",time[0]," time[rates_total-1] = ",time[rates_total-1
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Running Programs, Client Terminal Events, Working with Events

Variables

Declaring Variables

Variables must be declared before they are used. Unique names are used to identify variables. To declare a variable, you must specify its type and a unique name. Declaration of variable is not an operator.

Simple types are:

- char, short, int, long, uchar, ushort, uint, ulong integers;
- color integer representing the RGB-color;
- datetime the date and time, an unsigned integer containing the number of seconds since 0 hour January 1, 1970;
- bool boolean values *true* and *false*;
- double double-precision floating point number;
- float single-precision floating point number;
- string character strings.

Examples:

```
string szInfoBox; int    nOrders;
double dSymbolPrice;
bool   bLog;
datetime tBegin_Data   = D'2004.01.01 00:00';
color    cModify_Color = C'0x44,0xB9,0xE6';
```

Complex or compound types:

Structures are composite data types, constructed using other types.

```
struct MyTime
  {
   int hour;    // 0-23
   int minute;  // 0-59
   int second;  // 0-59
  };
...
MyTime strTime; // Variable of the previously declared structure MyTime
```

You can't declare variables of the structure type until you declare the structure.

Arrays

Array is the indexed sequence of identical-type data:

```
int    a[50];       // One-dimensional array of 50 integers.
double m[7][50];    // Two-dimensional array of seven arrays,
                    // each of them consisting of 50 numbers.
MyTime t[100];      // Array containing elements such as MyTime
```

Only an integer can be an array index. No more than four-dimensional arrays are allowed. Numbering of array elements starts with 0. The last element of a one-dimensional array has the number which is 1 less than the array size. This means that call for the last element of an array consisting of 50 integers will appear as a[49]. The same concerns multidimensional arrays: A dimension is indexed from 0 to the dimension size-1. The last element of a two-dimensional array from the example will appear as m[6][49].

Static arrays can't be represented as timeseries, i.e., the ArraySetAsSeries() function, which sets access to array elements from the end to beginning, can't be applied to them. If you want to provide access to an array the same as in timeseries, use the dynamic array object.

If there is an attempt to access out of the array range, the executing subsystem will generate a critical error and the program will be stopped.

Access Specifiers

Access specifiers define how the compiler can access variables, members of structures or classes.

The const specifier declares a variable as a constant, and does not allow to change this variable during runtime. A single initialization of a variable is allowed when declaring it.

Example:

```
int OnCalculate (const int rates_total,      // size of input time series
                 const int prev_calculated,  // bars handled in previous c
                 const datetime& time[],     // Time
                 const double& open[],       // Open
                 const double& high[],       // High
                 const double& low[],        // Low
                 const double& close[],      // Close
                 const long& tick_volume[],  // Tick Volume
                 const long& volume[],       // Real Volume
                 const int& spread[]         // Spread
   );
```

To access members of structures and classes use the following qualifiers:

· public allows unrestricted access to the variable or class method

· protected allows access from methods of this class, as well as from methods of publicly inherited classes. Other access is impossible;

- private allows access to variables and class methods only from methods of the same class.
- virtual applies only to class methods (but not to methods of structures) and tells the compiler that this method should be placed in the table of virtual functions of the class.

Storage Classes

There are three storage classes: static, input and extern. These modifiers of a storage class explicitly indicate to the compiler that corresponding variables are distributed in a pre-allocated area of memory, which is called the global pool. Besides, these modifiers indicate the special processing of variable data. If a variable declared on a local level is not a static one, memory for such a variable is allocated automatically at a program stack. Freeing of memory allocated for a non-static array is also performed automatically when going beyond the visibility area of the block, in which the array is declared.

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects, Static Members of a Class

Local Variables

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

Example:

```
int somefunc()   {
   int ret_code=0;
   ...
   return(ret_code);
  }
```

Scope of a variable is a program part, in which a variable can be referred to. Variables declared inside a block (at the internal level), have the block as their scope. The block scope start with the variable declaration and ends with the final right brace.

Local variables declared in the beginning of a function also have the scope of block, as well as function parameters that are local variables. Any block can contain variable declarations. If blocks are nested and the identifier in the external block has the same name as the identifier in the internal block, the external block identifier is hidden, until the operation of the internal block is over.

Example:

```
void OnStart()
  {
//---
   int i=5;      // local variable of the function
     {
      int i=10;  // function variable
      Print("Inside block i = ",i); // result is  i=10;
     }
   Print("Outside block i = ",i);  // result is  i=5;
  }
```

This means that while the internal block is running, it sees values of its own local identifiers, not the values of identifiers with identical names in the external block.

Example:

```
void OnStart()
  {
//---
   int i=5;      // local variable of the function
   for(int i=0;i<3;i++)
      Print("Inside for i = ",i);
   Print("Outside the block i = ",i);
  }
/* Execution result
Inside for i = 0
Inside for i = 1
Inside for i = 2
Outside block i = 5
*/
```

Local variables declared as static have the scope of the block, despite the fact that they exist since the program start.

Stack

In every MQL4 program, a special memory area called stack is allocated for storing local function variables that are created automatically. One stack is allocated for all functions. The default stack size is 256 kb, the stack size can be managed using the #property stacksize compiler directive.

Static local variables are stored in the same place where other static and global variables are stored - in a special memory area, which exists separately from the stack. Dynamically created variables also use a memory area separate from the stack.

With each function call, a place on the stack is allocated for internal non-static variables. After exiting the function, the memory is available for use again.

If from the first function the second one is called, then the second function occupies the required size from the remaining stack memory for its variables. Thus, when using included functions, stack memory will be sequentially occupied for each function. This may lead to a shortage of memory during one of the function calls, such a situation is called stack overflow.

Therefore, for large local data you should better use dynamic memory - when entering a function, allocate the memory, which is required for local needs, in the system (new, ArrayResize()), and when exiting the function, release the memory (delete, ArrayFree()).

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of

Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Formal Parameters

Parameters passed to the function are local. The scope is the function block. Formal parameters must have names differing from those of external variables and local variables defined within one function. Some values can be assigned to formal parameters in the function block. If a formal parameter is declared with the const modifier, its value can't be changed within the function.

Example:

```
void func(const int & x[], double y, bool z)   {
   if(y>0.0 && !z)
      Print(x[0]);
   ...
  }
```

Formal parameters can be initialized by constants. In this case, the initializing value is considered as the default value. Parameters, next to the initialized one, must also be initialized.

Example:

```
void func(int x, double y = 0.0, bool z = true)
  {
   ...
  }
```

When calling such a function, the initialized parameters can be omitted, the defaults being substituted instead of them.

Example:

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

Parameters of simple types are passed by value, i.e., modifications of the corresponding local variable of this type inside the called function will not be reflected in the calling function. Arrays of any type and data of the structure type are always passed by reference. If it is necessary to prohibit modifying the array or structure contents, the parameters of these types must be declared with the *const* keyword.

There is an opportunity to pass parameters of simple types by reference. In this case, modification of such parameters inside the calling function will affect the corresponding variables passed by reference. In order to indicate that a parameter is passed by reference, put the & modifier after the data type.

Example:

```
void func(int& x, double& y, double & z[])
  {
   double calculated_tp;
   ...
   for(int i=0; i<OrdersTotal(); i++)
     {
      if(i==ArraySize(z))       break;
      if(OrderSelect(i)==false) break;
      z[i]=OrderOpenPrice();
     }
   x=i;
   y=calculated_tp;
  }
```

Parameters passed by reference can't be initialized by default values.

Maximum 64 parameters can be passed into a function.

See also

Input Variables, Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Static Variables

The storage class of static defines a static variable. The static modifier is indicated before the data type.

Example:

```
int somefunc()   {
   static int flag=10;
   ...
   return(flag);
  }
```

A static variable can be initialized by a constant or constant expression corresponding to its type, unlike a simple local variable, which can be initialized by any expression.

Static variables exist from the moment of program execution and are initialized only once after the program is loaded. If the initial values are not specified, variables of the static storage class are taking zero initial values. The scope of the static variables is the same as the scope of the global variables: the lifetime of the mql4-program. The scope of a static variable is local to the block in which the variable is defined.

Local variables declared with the *static* keyword retain their values throughout the function lifetime. With each next function call, such local variables contain the values that they had during the previous call.

Any variables in a block, except formal parameters of a function, can be defined as static. If a variable declared on a local level is not a static one, memory for such a variable is allocated automatically at a program stack.

Example:

```
int Counter()
  {
   static int count;
   count++;
   if(count%100==0) Print("Function Counter has been called ",count," time
   return count;
  }
void OnStart()
  {
//---
   int c=345;
   for(int i=0;i<1000;i++)
     {
      int c=Counter();
     }
   Print("c =",c);
  }
```

See also

Data Types, Encapsulation and Extensibility of Types, Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects, Static Class Members

Global Variables

Global variables are created by placing their declarations outside function descriptions. Global variables are defined at the same level as functions, i.e., they are not local in any block.

Example:

```
int GlobalFlag=10;   // Global variable int OnStart()
  {
   ...
  }
```

The scope of global variables is the entire program. Global variables are accessible from all functions defined in the program. They are initialized to zero unless another initial value is explicitly defined. A global variable can be initialized only by a constant or constant expression that corresponds to its type.

Global variables are initialized only once after the program is loaded into the client terminal memory and before the first handling of the Init event. For global variables representing class objects, during their initialization the corresponding constructors are called.

The scope of the global variables is the same as the scope of the static variables : the lifetime of MQL4 program.

Note: Variables declared at global level must not be mixed up with the client terminal global variables that can be accessed using the GlobalVariable...() functions.

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

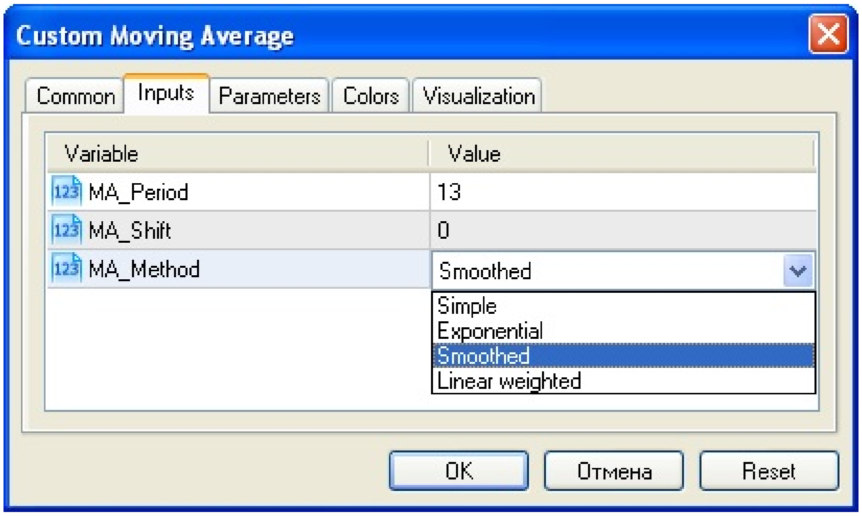
Input Variables

The input storage class defines the external variable. The *input* modifier is indicated before the data type. A variable with the input modifier can't be changed inside mql4-programs, such variables can be accessed for reading only. Values of input variables can be changed only by a user from the program properties window. External variables are always reinitialized immediately before the OnInit() is called.

Example:

```
//--- input parameters input int                   MA_Period=13;
input int                 MA_Shift=0;
input ENUM_MA_METHOD MA_Method=MODE_SMMA;
```

Input variables determine the input parameters of a program. They are available from the Properties window of a program.

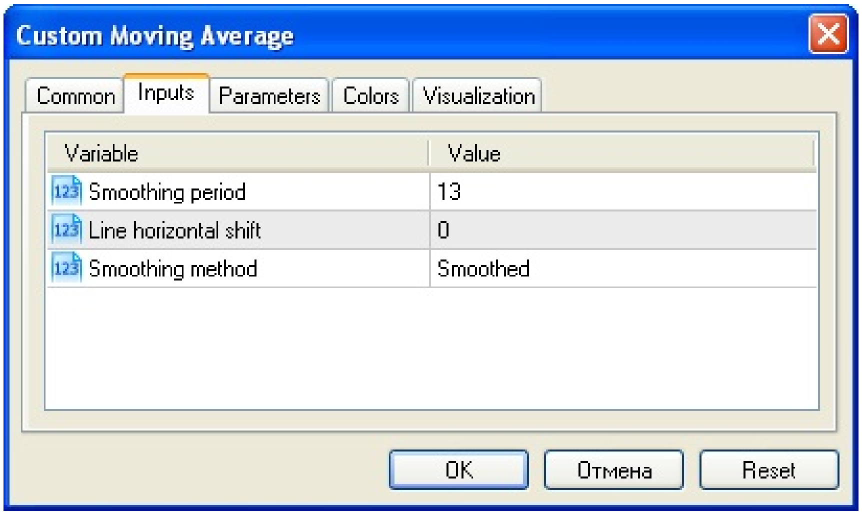


Variable	Value
MA_Period	13
MA_Shift	0
MA_Method	Smoothed

There is another way to set how your input parameter will look like in the Inputs tab. For this, place a string comment after the description of an input parameter in the same line. In this way you can make names of input parameters more understandable for users.

Example:

```
//--- input parameters
input int                 InpMAPeriod=13;          // Smoothing period
input int                 InpMAShift=0;            // Line horizontal shift
input ENUM_MA_METHOD InpMAMethod=MODE_SMMA;   // Smoothing method
```



Variable	Value
Smoothing period	13
Line horizontal shift	0
Smoothing method	Smoothed

Note: Arrays and variables of complex types can't act as input variables.

Note: The length of a string comment for Input variables cannot exceed 63 characters.

Passing Parameters When Calling Custom Indicators from MQL4 Programs

Custom Indicators are called using the iCustom() function. After the name of the custom indicator, parameters should go in a strict accordance with the declaration of input variables of this custom indicator. If indicated parameters are less than input variables declared in the called custom indicator, the missing parameters are filled with values specified during the declaration of variables.

If the custom indicator uses the OnCalculate function of the first type (i.e., the indicator is calculated using the same array of data), then one of ENUM_APPLIED_PRICE values or handle of another indicator should be used as the last parameter when calling such a custom indicator. All parameters corresponding to input variables must be clearly indicated.

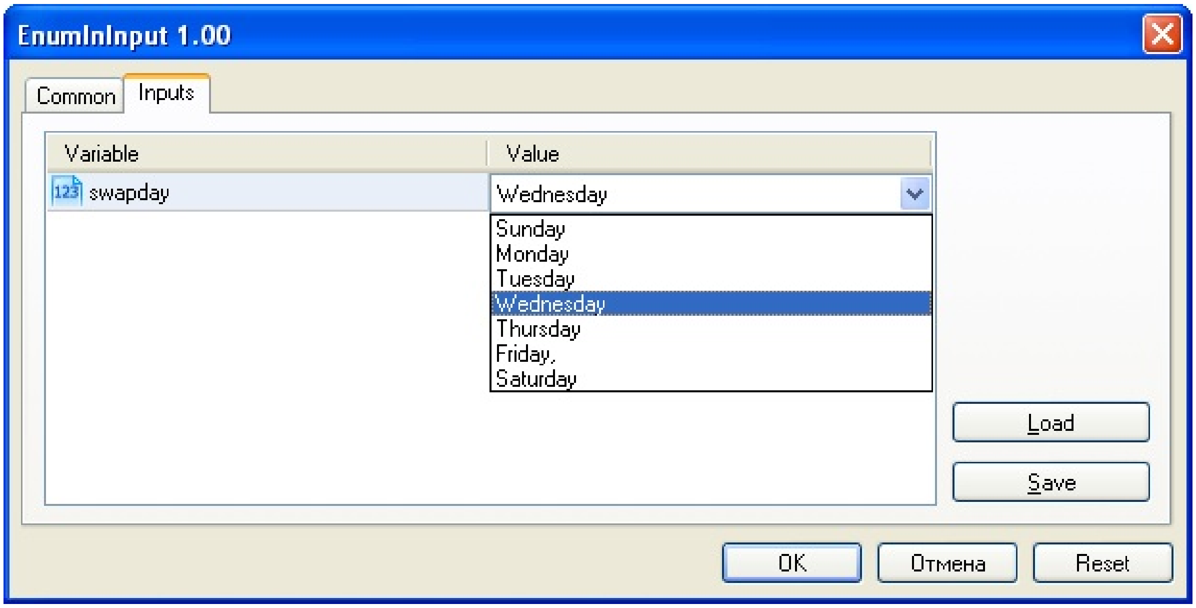
Enumerations as input Parameters

Not only built-in enumerations provided in MQL4, but also user defined variables can be used as input variables (input parameters for mql4 programs). For example, we can create the dayOfWeek enumeration, describing days of the week, and use the input variable to specify a particular day of the week, not as a number, but in a more common way.

Example:

```
#property script_show_inputs
//--- day of week
enum dayOfWeek
  {
   S=0,     // Sunday
   M=1,     // Monday
   T=2,     // Tuesday
   W=3,     // Wednesday
   Th=4,    // Thursday
   Fr=5,    // Friday,
   St=6,    // Saturday
  };
//--- input parameters
input dayOfWeek swapday=W;
```

In order to enable a user to select a necessary value from the properties window during the script startup, we use the preprocessor command #property script_show_inputs. We start the script and can choose one of values of the dayOfWeek enumeration from the list. We start the EnumInInput script and go to the Inputs tab. By default, the value of swapday (day of triple swap charge) is Wednesday (W = 3), but we can specify any other value, and use this value to change the program operation.


EnumInInput 1.00

Variable	Value
swapday	Wednesday

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday,
Saturday

Number of possible values of an enumeration is limited. In order to select an input value the drop-down list is used. Mnemonic names of enumeration members are used for values displayed in the list. If a comment is associated with a mnemonic name, as shown in this example, the comment content is used instead of the mnemonic name.

Each value of the dayOfWeek enumeration has its value from 0 to 6, but in the list of parameters, comments specified for each value will be shown. This provides additional flexibility for writing programs with clear descriptions of input parameters.

Variables with sinput Modifier

Variables with input modifier allow not only setting external parameters values when launching programs but are also necessary when optimizing trading strategies in the Strategy Tester. Each input variable excluding the one of a string type can be used in optimization.

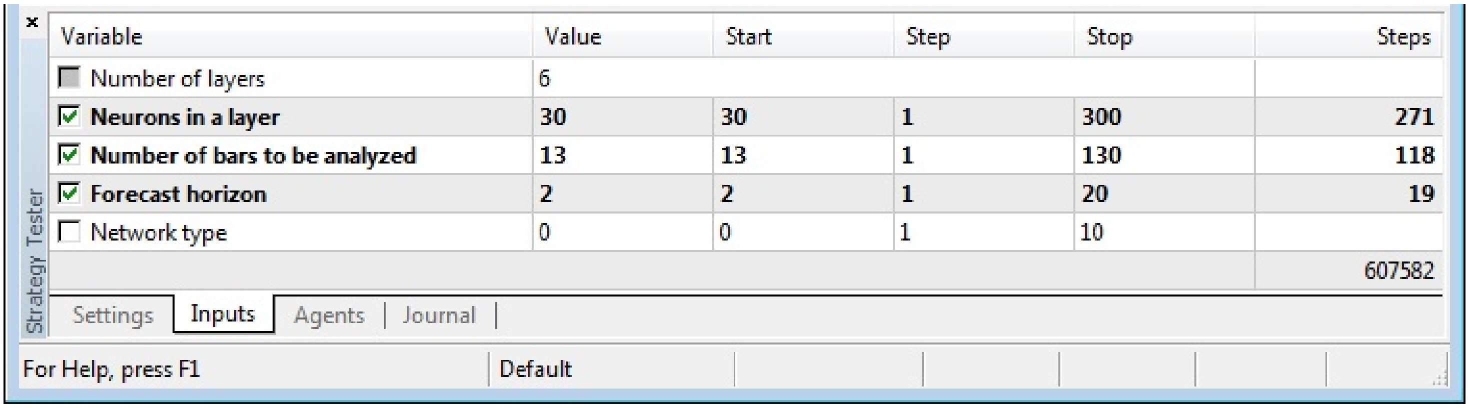
Sometimes, it is necessary to exclude some external program parameters from the area of all passes in the tester. sinput memory modifier has been introduced for such cases. sinput stands for static external variable declaration (sinput = static input). It means that the following declaration in an Expert Advisor code

```
sinput        int layers=6;   // Number of layers
```

will be equivalent to the full declaration

```
static input int layers=6;   // Number of layers
```

The variable declared with sinput modifier is an input parameter of MQL4 program. The value of this parameter can be changed when launching the program. However, this variable is not used in the optimization of input parameters. In other words, its values are not enumerated when searching for the best set of parameters fitting a specified condition.



Variable	Value	Start	Step	Stop	Steps
☐ Number of layers	6				
☑ **Neurons in a layer**	**30**	**30**	**1**	**300**	**271**
☑ **Number of bars to be analyzed**	**13**	**13**	**1**	**130**	**118**
☑ **Forecast horizon**	**2**	**2**	**1**	**20**	**19**
☐ Network type	0	0	1	10	
					607582

The Expert Advisor shown above has 5 external parameters. "Number of layers" is declared to be sinput and equal to 6. This parameter cannot be changed during a trading strategy optimization. We can specify the necessary value for it to be used further on. Start, Step and Stop fields are not available for such a variable.

Therefore, users will not be able to optimize this parameter after we specify sinput modifier for the variable. In other words, the terminal users will not be able to set initial and final values for it in the Strategy Tester for automatic enumeration in the specified range during optimization.

However, there is one exception to this rule: sinput variables can be varied in optimization tasks using ParameterSetRange() function. This function has been introduced specifically for the program control of available values sets for any input variable including the ones declared as static input (sinput). The ParameterGetInput() function allows to receive input variables values when optimization is launched (in OnTesterInit() handler) and to reset a change step value and a range, within which an optimized parameter values will be enumerated.

In this way, combining the sinput modifier and two functions that work with input parameters, allows to create a flexible rules for setting optimization intervals of input parameters that depend on values of another input parameters.

See also

iCustom, Enumerations, Properties of Programs

Extern variables

The extern storage class defines the external variable. The *extern* modifier is indicated before the data type.

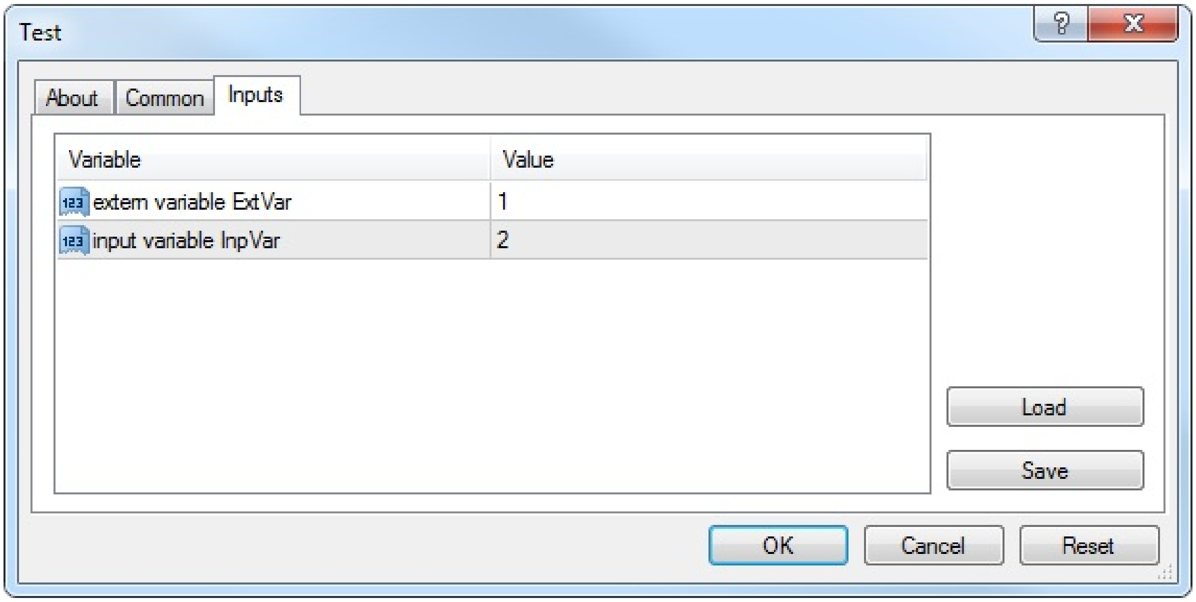
```
//--- extern parameters extern int            MA_Period=13;
extern int            MA_Shift=0;
extern ENUM_MA_METHOD MA_Method=MODE_SMMA;
```

Similar to input-variables, extern ones also determine the input parameters of an mql4 program. They are available from the Properties window. Unlike input variables, values of extern variables can be modified in the program during its operation. External variables are always reinitialized immediately before the OnInit() is called.

Example:

```
//--- strict compilation mode
#property strict
//--- show input parameters
#property show_inputs
//--- declare extern and input variables
extern int ExtVar=1;   // ExtVar extern variable
input  int InpVar=2;   // InpVar input variable
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- display the values of ExtVar and InpVar variables
   PrintFormat("Extern=%d, Input=%d",ExtVar,InpVar);
//--- increase the value of ExtVar variable by one
   ExtVar++;
//--- attempt to change the input variable will result in the compilation
//--- InpVar++;
//--- display the values of ExtVar and InpVar variables
   PrintFormat("Extern=%d, Input=%d",ExtVar,InpVar);
  }
```

Strict compilation mode with the output of the input parameters window is set in this script. Therefore, the values set in the string comments instead of ExtVar and InpVar variable names are displayed in Variable field.



Variable	Value
extern variable ExtVar	1
input variable InpVar	2

Note: Arrays and variables of complex types can't act as extern variables.

Note: The length of a string comment for extern variables cannot exceed 63 characters.

See also

Input Variables, Data Types, Encapsulation and Extensibility of Types, Initialization of Variables, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Initialization of Variables

Any variable can be initialized during definition. If a variable is not initialized explicitly, the value stored in this variable can be any. Implicit initialization is not used.

Global and static variables can be initialized only by a constant of the corresponding type or a constant expression. Local variables can be initialized by any expression, not just a constant.

Initialization of global and static variables is performed only once. Initialization of local variables is made every time you call the corresponding functions.

Examples:

```
int    n       = 1; string s       = "hello";
double f[]     = { 0.0, 0.236, 0.382, 0.5, 0.618, 1.0 };
int    a[4][4] = { {1, 1, 1, 1}, {2, 2, 2, 2}, {3, 3, 3, 3}, {4, 4, 4, 4 }
//--- from tetris
int    right[4]={WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER,
                 WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER};
//--- initialization of all fields of the structure with zero values
MqlTradeRequest request={0};
```

List of values of the array elements must be enclosed in curly brackets. Missed initializing sequences are considered equal to 0. The initializing sequence must have at least one value: this value is initialized to the first element of the corresponding structure or array, missing elements are considered equal to zero.

If the size of the initialized array is not specified, it is determined by a compiler, based on the size of the initialization sequence. Multi-dimensional arrays cannot be initialized by a one-dimensional sequence (a sequence without additional curly brackets), except for the case, when only one initializing element is specified (zero, as a rule).

Arrays (including those announced at the local level) can be initialized only by constants.

Examples:

```
struct str3
  {
   int               low_part;
   int               high_part;
  };
struct str10
  {
   str3              s3;
   double            d1[10];
   int               i3;
  };
void OnStart()
  {
   str10 s10_1={{1,0},{1.0,2.1,3.2,4.4,5.3,6.1,7.8,8.7,9.2,10.0},100};
   str10 s10_2={{1,0},{0},100};
   str10 s10_3={{1,0},{1.0}};
//---
   Print("1.  s10_1.d1[5] = ",s10_1.d1[5]);
   Print("2.  s10_2.d1[5] = ",s10_2.d1[5]);
   Print("3.  s10_3.d1[5] = ",s10_3.d1[5]);
   Print("4.  s10_3.d1[0] = ",s10_3.d1[0]);
  }
```

For structure type variable partial initialization is allowed, as well as for static arrays (with an implicitly set size). You can initialize one or more first elements of a structure or array, the other elements will be initialized with zeroes in this case.

See also

Data Types, Encapsulation and Extensibility of Types, Visibility Scope and Lifetime of Variables, Creating and Deleting Objects

Visibility Scope and Lifetime of Variables

There are two basic types of scope: local scope and global scope.

A variable declared outside all functions is located into the global scope. Access to such variables can be done from anywhere in the program.These variables are located in the global pool of memory, so their lifetime coincides with the lifetime of the program.

A variable declared inside a block (part of code enclosed in curly brackets) belongs to the local scope. Such a variable is not visible (and therefore not available) outside the block, in which it is declared. The most common case of local declaration is a variable declared within a function. A variable declared locally, is located on the stack, and the lifetime of such a variable is equal to the lifetime of the function.

Since the scope of a local variable is the block in which it is declared, it is possible to declare variables with the same name, as those of variables declared in other blocks; as well as of those declared at upper levels, up to the global level.

Example:

```
void CalculateLWMA(int rates_total,int prev_calculated,int begin,const dou
   int        i,limit;
   static int weightsum=0;
   double     sum=0;
//---
   if(prev_calculated==0)
     {
      limit=MA_Period+begin;
      //--- set empty value for first limit bars
      for(i=0; i<limit; i++) LineBuffer[i]=0.0;
      //--- calculate first visible value
      double firstValue=0;
      for(int i=begin; i<limit; i++)
        {
         int k=i-begin+1;
         weightsum+=k;
         firstValue+=k*price[i];
        }
      firstValue/=(double)weightsum;
      LineBuffer[limit-1]=firstValue;
     }
   else
     {
      limit=prev_calculated-1;
     }

   for(i=limit;i<rates_total;i++)
     {
      sum=0;
      for(int j=0; j<MA_Period; j++) sum+=(MA_Period-j)*price[i-j];
      LineBuffer[i]=sum/weightsum;
     }
//---
  }
```

Pay attention to the variable i, declared in line

```
      for(int i=begin; i<limit; i++)
        {
         int k=i-begin+1;
         weightsum+=k;
         firstValue+=k*price[i];
        }
```

Its scope is only the for loop; outside of this loop there is another variable with the same name, declared at the beginning of the function. In addition, the k variable is declared in the loop body, its scope is the loop body.

Local variables can be declared with the access specifier static. In this case, the compiler has a variable in the global pool of memory. Therefore, the lifetime of a static variable is equal to the lifetime of the program. Here the scope of such a variable is limited to the block in which it is declared.

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Creating and Deleting Objects

Creating and Deleting Objects

After a MQL4 program is loaded for execution, memory is allocated to each variable according to its type. According to the access level, all variables are divided into two types - global variables and local variables. According to the memory class, they can be input parameters of a MQL4 program, static and automatic. If necessary, each variable is initialized by a corresponding value. After being used a variable is unintialized and memory used by it is returned to the MQL4 executable system.

Initialization and Deinitialization of Global Variables

Global variables are initialized automatically right after a MQL4 program is loaded and before any of function is called. During initialization initial values are assigned to variables of simple types and a constructor (if there is any) is called for objects. Input variables are always declared at a global level, and are initialized by values set by a user in the dialog during the program start.

Despite the fact that static variables are usually declared at a local level, the memory for these variables is pre-allocated, and initialization is performed right after a program is loaded, the same as for global variables.

The initialization order corresponds to the variable declaration order in the program. Deinitialization is performed in the reverse order. This rule is true only for the variables that were not created by the new operator. Such variables are created and initialized automatically right after loading, and are deinitialized before the program unloading.

Initialization and Deinitialization of Local Variables

If a variable declared on a local level is not a static one, memory is allocated automatically for such a variable. Local variables, as well as global ones, are initialized automatically at the moment when the program execution meets their declaration. Thus the initialization order corresponds to the order of declaration.

Local variables are deinitialized at the end of the program block, in which they were declared, and in the order opposite to their declaration. A program block is a compound operator that can be a part of selection operator switch, loop operator (for, while, do-while), a function body or a part of the if-else operator.

Local variables are initialized only at the moment when the program execution meets the variable declaration. If during the program execution the block, in which the variable is declared, was not executed, such a variable is not initialized.

Initialization and Deinitialization of Objects Placed

A special case is that with object pointers, because declaration of a pointer does not entail initialization of a corresponding objects. Dynamically placed objects are initialized only at the moment when the class sample is created by the new operator. Initialization of objects presupposes call of a constructor of a corresponding class. If there is no corresponding constructor in the class, its members of a simple type will not be automatically initialized; members of types string, dynamic array and complex object will be automatically initialized.

Pointers can be declared on a local or global level; and they can be initialized by the empty value of NULL or by the value of the pointer of the same or inherited type. If the *new* operator is called for a pointer declared on a local level, the *delete* operator for this pointer must be performed before exiting the level. Otherwise the pointer will be lost and the explicit deletion of the object will fail.

All objects created by the expression of *object_pointer*=new *Class_name*, must be then deleted by the delete(*object_pointer*) operator. If for some reasons such a variable is not deleted by the delete operator when the program is completed, the corresponding entry will appear in the "Experts" journal. One can declare several variables and assign a pointer of one object to all of them.

If a dynamically created object has a constructor, this constructor will be called at the moment of the *new* operator execution. If an object has a destructor, it will be called during the execution of the *delete* operator.

Thus dynamically placed objects are created only at the moment when the corresponding *new* operator is invoked, and are assuredly deleted either by the *delete* operator or automatically by the executing system of MQL4 during the program unloading. The order of declaration of pointers of dynamically created object doesn't influence the order of their initialization. The order of initialization and deinitialization is fully controlled by the programmer.

Dynamic memory allocation in MQL4

When working with dynamic arrays, released memory is immediately returned back to the operating system.

When working with dynamic class objects using the new operator, first memory is requested from the class memory pool the memory manager is working with. If there is not enough memory in the pool, memory is requested from the operating system. When deleting the dynamic object using the delete operator, released memory is immediately returned back to the class memory pool.

Memory manager releases memory back to the operating system immediately after exiting the following event handling functions: OnInit(), OnDeinit(), OnStart(), OnTick(), OnCalculate(), OnTimer(), OnTester(), OnChartEvent().

Brief Characteristics of Variables

The main information about the order of creation, deletion, about calls of constructors and destructors is given in the below table.

	Global automatic variable	Local automatic variable	Dynamically created object
Initialization	right after a mql4 program is loaded	when the code line where it is declared is reached during execution	at the execution of the **new** operator
Initialization order	in the order of declaration	in the order of declaration	irrespective of the order of declaration
Deinitialization	before a mql4 program is unloaded	when execution exits the declaration block	when the **delete** operator is executed or before a mql4 program is unloaded
Deinitialization order	in the order opposite to the initialization order	in the order opposite to the initialization order	irrespective of the initialization order
Constructor call	at mql4 program loading	at initialization	at the execution of the *new* operator
Destructor call	at mql4 program unloading	when exiting the block where the variable was initialized	at the execution of the *delete* operator
Error logs	log message in the "Experts" journal about the attempt to delete an automatically	log message in the "Experts" journal about the attempt to delete an automatically	log message in the "Experts" journal about undeleted dynamically created objects at the unload of a mql4

	created object	created object	program

See also

Data Types, Encapsulation and Extensibility of Types,Initialization of Variables, Visibility Scope and Lifetime of Variables

Preprocessor

Preprocessor is a special subsystem of the MQL4 compiler that is intended for preparation of the program source code immediately before the program is compiled.

Preprocessor allows enhancement of the source code readability. The code can be structured by including of specific files containing source codes of mql4-programs. The possibility to assign mnemonic names to specific constants contributes to enhancement of the code readability.

Preprocessor also allows determining specific parameters of mql4-programs:

· Declare constants

· Set program properties

· Include files in program text

· Import functions

· Conditional compilation

If the # symbol is used as the first character in a line of the program, this line is considered as a preprocessor directive. A preprocessor directive ends with a line feed character.

Macro substitution (#define, #undef)

The #define directive can be used to assign mnemonic names to constants. There are two forms:

```
#define identifier expression                  // parameter-free form #de
```

The #define directive substitutes **expression** for all further found entries of *identifier* in the source text. The *identifier* is replaced only if it is a separate token. The *identifier* is not replaced if it is part of a comment, part of a string, or part of another longer identifier.

The constant identifier is governed by the same rules as variable names. The value can be of any type:

```
#define ABC               100
#define PI                3.14
#define COMPANY_NAME      "MetaQuotes Software Corp."
...
void ShowCopyright()
  {
   Print("Copyright  2001-2013, ",COMPANY_NAME);
   Print("http://www.metaquotes.net");
  }
```

expression can consist of several tokens, such as keywords, constants, constant and non-constant expressions. **expression** ends with the end of the line and can't be transferred to the next line.

Example:

```
#define TWO        2
#define THREE      3
#define INCOMPLETE TWO+THREE
#define COMPLETE  (TWO+THREE)
void OnStart()
  {
   Print("2 + 3*2 = ",INCOMPLETE*2);
   Print("(2 + 3)*2 = ",COMPLETE*2);
  }
// Result
// 2 + 3*2 = 8
// (2 + 3)*2 = 10
```

Parametric Form #define

With the parametric form, all the subsequent found entries of identifier will be replaced by expression taking into account the actual parameters. For example:

```
 // example with two parameters a and b
#define A 2+3
#define B 5-1
#define MUL(a, b) ((a)*(b))

double c=MUL(A,B);
Print("c=",c);
/*
expression double c=MUL(A,B);
is equivalent to double c=((2+3)*(5-1));
*/
// Result
// c=20
```

Be sure to enclose parameters in parentheses when using the parameters in expression, as this will help avoid non-obvious errors that are hard to find. If we rewrite the code without using the brackets, the result will be different:

```
 // example with two parameters a and b
#define A 2+3
#define B 5-1
#define MUL(a, b) a*b

double c=MUL(A,B);
Print("c=",c);
/*
expression double c=MUL(A,B);
is equivalent to double c=2+3*5-1;
*/
// Result
// c=16
```

When using the parametric form, maximum 8 parameters are allowed.

```
 // correct parametric form
#define LOG(text)  Print(__FILE__,"(",__LINE__,") :",text)   // one paramet

 // incorrect parametric form
#define WRONG_DEF(p1, p2, p3, p4, p5, p6, p7, p8, p9)   p1+p2+p3+p4 // mor
```

The #undef directive

The #undef directive cancels declaration of the macro substitution, defined before.

Example:

```
#define MACRO

void func1()
  {
#ifdef MACRO
   Print("MACRO is defined in ",__FUNCTION__);
#else
   Print("MACRO is not defined in ",__FUNCTION__);
#endif
  }

#undef MACRO

void func2()
  {
#ifdef MACRO
   Print("MACRO is defined in ",__FUNCTION__);
#else
   Print("MACRO is not defined in ",__FUNCTION__);
#endif
  }

void OnStart()
  {
   func1();
   func2();
  }

/* Result:
 MACRO is defined in func1
 MACRO is not defined in func2
*/
```

See also

Identifiers, Character Constants

Program Properties (#property)

Every MQL4-program allows to specify additional specific parameters named #property that help client terminal in proper servicing for programs without the necessity to launch them explicitly. This concerns external settings of indicators, first of all. Properties described in included files are completely ignored. Properties must be specified in the main mq4 file.

```
#property identifier value
```

The compiler will write declared values in the configuration of the module executed.

Constant	Type	Description
strict		Compiler directive for strict compilation mode (see Updated MQL4)
icon	string	Path to the file of an image that will be used as an icon of the EX4 program. Path specification rules are the same as for resources. The property must be specified in the main module with the MQL4 source code. The icon file must be in the ICO format.
link	string	Link to the company website
copyright	string	The company name
version	string	Program version, maximum 31 characters
description	string	Brief text description of a MQL4-program. Several *description* can be present, each of them describes one line of the text. The total length of all *description* can not exceed 511 characters including line feed.
stacksize	int	MQL4 program stack size. The stack of sufficient size is necessary when executing function recursive calls. When launching a script or an Expert Advisor on the chart, the stack of at least 8 MB is allocated. In case of indicators, the stack size is always fixed and equal to 1 MB. When a program is launched in the strategy tester, the stack of 8 MB is always allocated for it.

library		A library; no start function is assigned, functions with the export modifier can be imported in other MQL4-programs
indicator_chart_window		Show the indicator in the chart window
indicator_separate_window		Show the indicator in a separate window
indicator_height	int	Fixed height of the indicator subwindow in pixels (property INDICATOR_HEIGHT)
indicator_buffers	int	Number of buffers for indicator calculation
indicator_minimum	double	The bottom scaling limit for a separate indicator window
indicator_maximum	double	The top scaling limit for a separate indicator window
indicator_labelN	string	Sets a label for the N-th graphic series displayed in DataWindow
indicator_colorN	color	The color for displaying line N, where N is the number of graphic series; numbering starts from 1
indicator_widthN	int	Line thickness in graphic series, where N is the number of graphic series; numbering starts from 1
indicator_styleN	int	Line style in graphic series, specified by the values of ENUM_LINE_STYLE. N is the number of graphic series; numbering starts from 1
indicator_typeN	int	Type of indicator drawing style. N is the number of graphic series; numbering starts from 1
indicator_levelN	double	Horizontal level of N in a separate indicator window
indicator_levelcolor	color	Color of horizontal levels of the indicator
indicator_levelwidth	int	Thickness of horizontal levels of the indicator
indicator_levelstyle	int	Style of horizontal levels of the indicator
script_show_confirm		Display a confirmation window before running the script
script_show_inputs		Display a window with the properties before running the script and disable this confirmation window
tester_file	string	File name from <terminal_data_folder>\MQL4\Files\ to be sent

		to a virtual server
tester_indicator	string	Indicator file name from <terminal_data_folder>\MQL4\Indicators\ to be sent to a virtual server
tester_library	string	Library file name from <terminal_data_folder>\MQL4\Libraries\ to be sent to a virtual server

tester_file, tester_indicator and tester_library properties are necessary for describing the list of the files required for working in virtual hosting.

Indicator files called in iCustom() function with a fixed name, as well as all library files used in MQL4 programs are copied automatically during the migration.

Find out more about migration of programs in the article "How to Prepare a Trading Account for Migration to Virtual Hosting".

Sample code for moving files to a hosting

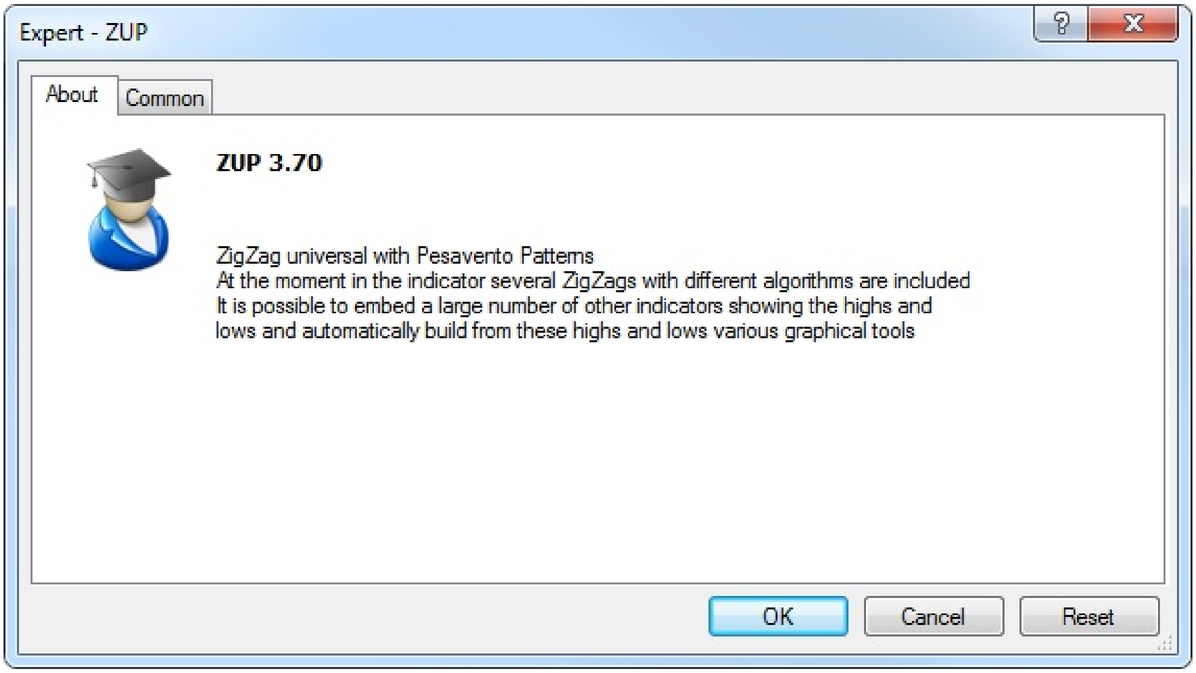
```
#property tester_file "trade_patterns.csv"    // file with the data to be
```

Example of an implicit indication of an indicator name in the code

```
string indicator_name="smoothed_ma.ex4";
double val=iCustom(NULL,0,indicator_name,13,1,0);
```

Sample Task of Description and Version Number

```
#property version     "3.70"      // Current version of the Expert Advisor
#property description "ZigZag universal with Pesavento Patterns"
#property description "At the moment in the indicator several ZigZags with
#property description "It is possible to embed a large number of other inc
#property description "lows and automatically build from these highs and l
```



ZUP 3.70

ZigZag universal with Pesavento Patterns
At the moment in the indicator several ZigZags with different algorithms are included
It is possible to embed a large number of other indicators showing the highs and
lows and automatically build from these highs and lows various graphical tools

Including Files (#include)

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

```
#include <file_name> #include "file_name"
```

Examples:

```
#include <WinUser32.mqh>
#include "mylib.mqh"
```

The preprocessor replaces the line *#include <file_name>* with the content of the file WinUser32.mqh. Angle brackets indicate that the WinUser32.mqh file will be taken from the standard directory (usually it is *terminal_installation_directory\MQL4\Include*). The current directory is not included in the search.

If the file name is enclosed in quotation marks, the search is made in the current directory (which contains the main source file). The standard directory is not included in the search.

See also

Importing Functions

Importing Function (#import)

Functions are imported from compiled MQL4 modules (*.ex4 files) and from operating system modules (*.dll files). The module name is specified in the *#import* directive. For compiler to be able to correctly form the imported function call and organize proper transmission parameters, the full description of functions is needed. Function descriptions immediately follow the *#import "module name"* directive. New command *#import* (can be without parameters) completes the block of imported function descriptions.

```
#import "file_name"     func1 define;
    func2 define;
    ...
    funcN define;
#import
```

Imported functions can have any names. Functions having the same names but from different modules can be imported at the same time. Imported functions can have names that coincide with the names of built-in functions. Operation of scope resolution defines which of the functions should be called.

The order of searching for a file specified after the #import keyword is described in Call of Imported Functions.

Since the imported functions are outside the compiled module, the compiler can not verify the validity of passed parameters. Therefore, to avoid run-time errors, one must accurately describe the composition and order of parameters passed to imported functions. Parameters passed to imported functions (both from EX4, and from the DLL-module) can have default values.

The following can't be used for parameters in imported functions:

· pointers (*);

· links to objects that contain dynamic arrays and/or pointers.

Classes, string arrays or complex objects that contain strings and/or dynamic arrays of any types cannot be passed as a parameter to functions imported from DLL.

Examples:

```
#import "user32.dll"
int    MessageBoxW(uint hWnd,string lpText,string lpCaption,uint uType);
#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);
#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);
#import
```

To import functions during execution of a mql4 program, early binding is used. This means that the library is loaded during the loading of a program using its ex4 program.

It's not recommended to use a fully qualified name of the loadable module of type *Drive:\Directory\FileName.Ext*. MQL4 libraries are loaded from the *terminal_dir\MQL4\Libraries* folder.

See also

Including Files

Conditional Compilation (#ifdef, #ifndef, #else, #endif)

Preprocessor conditional compilation directives allow compiling or skipping a part of the program depending on the fulfillment of a certain condition.

That condition can take one of the following forms.

```
#ifdef identifier   // the code located here is compiled if identifier ha
#endif
```

```
#ifndef identifier
   // the code located here is compiled if identifier is not currently def
#endif
```

Any of the conditional compilation directives can be followed by any number of lines possibly containing #else directive and ending with #endif. If the verified condition is true, the lines between #else and #endif are ignored. If the verified condition is not fulfilled, all lines between checking and #else directive (or #endif directive if the former is absent) are ignored.

Example:

```
#ifndef TestMode
   #define TestMode
#endif
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   #ifdef TestMode
      Print("Test mode");
   #else
      Print("Normal mode");
   #endif
  }
```

Depending on the program type and compilation mode, the standard macros are defined the following way:

__MQL4__ macro is defined when compiling *.mq4 file, __MQL5__ macro is defined when compiling *.mq5 one.
_DEBUG macro is defined when compiling in debug mode.
_RELEASE macro is defined when compiling in release mode.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   #ifdef __MQL5__
      #ifdef _DEBUG
         Print("Hello from MQL5 compiler [DEBUG]");
      #else
        #ifdef _RELEASE
           Print("Hello from MQL5 compiler [RELEASE]");
        #endif
     #endif
   #else
      #ifdef __MQL4__
         #ifdef _DEBUG
           Print("Hello from MQL4 compiler [DEBUG]");
        #else
           #ifdef _RELEASE
              Print("Hello from MQL4 compiler [RELEASE]");
           #endif
        #endif
     #endif
   #endif
  }
```

Object-Oriented Programming

Object-oriented programming (OOP) is programming primarily focused on data, while data and behavior are being inseparably linked. Data and behavior together constitute a class, while objects are class instances.

The components of the object-oriented approach are:

- Encapsulation and type extensibility
- Inheritance
- Polymorphism
- Overloading
- Virtual functions

OOP considers computation as modeling of behavior. The modeled item is the object represented by computational abstractions. Suppose we want to write a well known game "Tetris". To do this, we must learn how to model the appearance of random shapes composed of four squares joined together by edges. Also we need to regulate the falling speed of shapes, define operations of rotation and shift of shapes. Moving of shapes on the screen is limited by the well's boundaries, this requirement must also be modeled. Besides that, filled rows of cubes must be destroyed and achieved points must be counted.

Thus, this easy-to-understand game requires the creation of several models - shape model, well model, shape movement model and so on. All these models are abstractions, represented by calculations in the computer. To describe these models, the concept of Abstract Data Type, ADT (or complex data type) is used. Strictly speaking, the model of the "shapes" motion in the DOM is not a data type, but it is a set of operations on the "shape" data type, using the restrictions of the "well" data type.

Objects are class variables. Object-oriented programming allows you to easily create and use ADT. Object-oriented programming uses the inheritance mechanism. The benefit of inheritance is in the fact that it allows obtaining derivative types from data types already defined by a user.

For example, to create Tetris shapes, it's convenient to create a base class Shape first. The other classes representing all seven possible shape types can be derived on its basis. Behavior of shapes is defined in the base class, while implementation of behavior of each separate shape is defined in derivative classes.

In OOP objects are responsible for their behavior. ADT developer should include a code to describe any behavior that would normally be expected from the corresponding objects. The fact that the object itself is responsible for its behavior, greatly simplifies the task of programming for the user of this object.

If we want to draw a shape on the screen, we need to know where the center will be and how to draw it. If a separate shape knows how to draw itself, the programmer should send a "draw" message when using such a shape.

The MQL4 Language is a C++ like, and it also has the encapsulation mechanism for the implementation of ADT. On the one hand encapsulation combines the internal details of the implementation of a particular type, and on the other hand it combines externally accessible functions that can influence objects of this type. Implementation details may be inaccessible for a program that uses this type.

The concept of OOP has a set of related concepts, including the following:

· Simulation of actions from the real world

· User-defined data types

· Hiding the implementation details

· Possibility of the code reuse through inheritance

· Interpretation of function calls during execution

Some of these concepts are rather vague, some are abstract, others are general.

Encapsulation and Extensibility of Types

OOP is a balanced approach to writing software. Data and behavior are packed together. This encapsulation creates user-defined data types, extending the language data types and interacting with them. Types extensibility is an opportunity to add to the language user-defined data types, which are also easy to use, as well as basic types.

An abstract data type, for example, a string, is a description of the ideal, well known behavior type.

The string user knows that the string operations, such as concatenation or print, have a certain behavior. Concatenation and print operations are called methods.

A certain implementation of ADT may have some restrictions, for example, strings can be limited in length. These limitations affect the behavior opened to all. At the same time, internal or private implementation details do not affect directly the way the user sees the object. For example, the string is often implemented as an array, while the internal base address of this array and its name are not essential for the user.

Encapsulation is the ability to hide the implementation details when the open interfaces to user-defined type is provided. In MQL4, as well as in C++, class and structure definitions (class and struct) are used for the encapsulation provisions in combination with access keywords private, protected and public.

The public keyword shows that access to the members that stand behind it is open without restrictions. Without this keyword, class members are locked by default. Private members are accessible only by member functions only of its class.

Protected class functions are available to class functions not only in its class, but also in its inheritor classes. Public class functions are available for any function within the scope of the class declaration. The protection makes possible to hide part of the class implementation, thus preventing unexpected changes in the structure of data. Access restriction or data hiding is a feature of the object-oriented programming.

Usually, class functions are protected and declared with the protected modifier, the reading and writing of the values are performed by using special so-called set-and get-methods that are defined by the public access modifier.

Example:

```
class CPerson   {
protected:
   string            m_name;                     // name
public:
   void              SetName(string n){m_name=n;}// sets name
   string            GetName(){return (m_name);} // returns name
  };
```

This approach offers several advantages. First, by function name we can understand what it does - sets or gets the value of a class member. Secondly, perhaps in the future we will need to change the type of the m_name variable in the CPerson class or in any of its derivative classes.

In this case, we'll need just to change the implementation of functions SetName() and GetName(), while objects of the CPerson class will be available for using in a program without any code changes because the user will not even know that the data type of m_name has changed.

Example:

```
struct Name
  {
   string               first_name;                 // name
   string               last_name;                  // last name
  };

class CPerson
  {
protected:
   Name                 m_name;                     // name
public:
   void                 SetName(string n);
   string               GetName(){return(m_name.first_name+" "+m_name.last_na
private:
   string               GetFirstName(string full_name);
   string               GetLastName(string full_name);
  };

void CPerson::SetName(string n)
  {
   m_name.first_name=GetFirstName(n);
   m_name.last_name=GetLastName(n);
  }

string CPerson::GetFirstName(string full_name)
  {
   int pos=StringFind(full_name," ");
   if(pos>0) StringSetCharacter(full_name,pos,0);
   return(full_name);
  }

string CPerson::GetLastName(string full_name)
  {
   string ret_string;
   int pos=StringFind(full_name," ");
   if(pos>0) ret_string=StringSubstr(full_name,pos+1);
   else      ret_string=full_name;
   return(ret_string);
  }
```

See also

Data Types

Inheritance

The characteristic feature of OOP is the encouragement of code reuse through inheritance. A new class is made from the existing, which is called the base class. The derived class uses the members of the base class, but can also modify and supplement them.

Many types are variations of the existing types. It is often tedious to develop a new code for each of them. In addition, the new code implies new errors. The derived class inherits the description of the base class, thus any re-development and re-testing of code is unnecessary. The inheritance relationships are hierarchical.

Hierarchy is a method that allows to copy the elements in all their diversity and complexity. It introduces the objects classification. For example, the periodic table of elements has gases. They possess to properties inherent to all periodic elements.

Inert gases constitute the next important subclass. The hierarchy is that the inert gas, such as argon is a gas, and gas, in its turn, is part of the system. Such a hierarchy allows to interpret behaviour of inert gases easily. We know that their atoms contain protons and electrons, that is true for all other elements.

We know that they are in a gaseous state at room temperature, like all the gases. We know that no gas from inert gas subclass enters usual chemical reaction with other elements, and it is a property of all inert gases.

Consider an example of the inheritance of geometric shapes. To describe the whole variety of simple shapes (circle, triangle, rectangle, square etc.), the best way is to create a base class (ADT), which is the ancestor of all the derived classes.

Let's create a base class CShape, which contains just the most common members describing the shape. These members describe properties that are characteristic of any shape - the type of the shape and main anchor point coordinates.

Example:

```
//--- The base class Shape class CShape
  {
protected:
   int       m_type;                   // Shape type
   int       m_xpos;                   // X - coordinate of the base point
   int       m_ypos;                   // Y - coordinate of the base point
public:
             CShape(){m_type=0; m_xpos=0; m_ypos=0;} // constructor
   void      SetXPos(int x){m_xpos=x;} // set X
   void      SetYPos(int y){m_ypos=y;} // set Y
  };
```

Next, create new classes derived from the base class, in which we will add necessary fields, each specifying a certain class. For the Circle shape it is necessary to add a member that contains the radius value. The Square shape is characterized by the side value. Therefore, derived classes, inherited from the base class CShape will be declared as follows:

```
//--- The derived class circle
class CCircle : public CShape        // After a colon we define the base c
  {                                    // from which inheritance is made
private:
   int                m_radius;        // circle radius

public:
                   CCircle(){m_type=1;}// constructor, type 1
  };
```

For the Square shape class declaration is similar:

```
//--- the derived class Square
class CSquare : public CShape        // After a colon we define the base c
  {                                    // from which inheritance is made
private:
   int              m_square_side;     // square side

public:
                  CSquare(){m_type=2;} // constructor, type 2
  };
```

It should be noted that while object is created the base class constructor is called first, and then the constructor of the derived class is called. When an object is destroyed first the destructor of the derived class is called, and then a base class destructor is called.

Thus, by declaring the most general members in the base class, we can add an additional members in derived classes, which specify a particular class.

Inheritance allows creating powerful code libraries that can be reused many times.

The syntax for creating a derived class from an already existing one is as follows:

```
class class_name :
          (public | protected | private) opt  base_class_name
  {
    class members declaration
  };
```

One of aspects of the derived class is the visibility (openness) of its members successors (heirs). The public, protected and private keywords are used to indicate the extent, to which members of the base class will be available for the derived one. The public keyword after a colon in the header of a derived class indicates that the protected and public members of the base class CShape should be inherited as protected and public members of the derived class CCircle.

The private class members of the base class are not available for the derived class. The public inheritance also means that derived classes (CCircle and CSquare) are CShapes. That is, the Square (CSquare) is a shape (CShape), but the shape does not necessarily have to be a square.

The derived class is a modification of the base class, it inherits the protected and public members of the base class. The constructors and destructors of the base class cannot be inherited. In addition to members of the base class, new members are added in a derivative class.

The derived class may include the implementation of member functions, different from the base class. It has nothing common with an overload, when the meaning of the same function name may be different for different signatures.

In protected inheritance, public and protected members of base class become protected members of derived class. In private inheritance, the public and protected members of base class become private members of the derived class.

In protected and private inheritance, the relation that "the object of a derivative class is object of a base class" is not true. The protected and private inheritance types are rare, and each of them needs to be used carefully.

It should be understood that the type of inheritance (public, protected or private) does not affect the ways of **accessing the members of base classes in the hierarchy of inheritance from a derived class**. With any type of inheritance, only base class members declared with public and protected access specifiers will be available out of the derived classes. Let's consider it in the following example:

```
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//+------------------------------------------------------------------+
//| Example class with a few access types                            |
//+------------------------------------------------------------------+
class CBaseClass
  {
private:             //--- The private member is not available from derive
   int               m_member;
protected:           //--- The protected method is available from the base
   int               Member(){return(m_member);}
public:              //--- Class constructor is available to all members c
                     CBaseClass(){m_member=5;return;};
private:             //--- A private method for assigning a value to m_mem
   void              Member(int value) { m_member=value;};

  };
//+------------------------------------------------------------------+
//| Derived class with errors                                        |
//+------------------------------------------------------------------+
class CDerived: public CBaseClass // specification of public inheritance c
  {
public:
   void Func() // In the derived class, define a function with calls to ba
     {
      //--- An attempt to modify a private member of the base class
      m_member=0;        // Error, the private member of the base class is
      Member(0);         // Error, the private method of the base class is
      //--- Reading the member of the base class
      Print(m_member);   // Error, the private member of the base class is
      Print(Member());   // No error, protected method is available from t
     }
  };
```

In the above example, CBaseClass has only a public method the constructor. Constructors are called automatically when creating a class object. Therefore, the private member m_member and the protected methods Member() cannot be called from the outside. But in case of public inheritance, the Member()

method of the base class will be available from the derived classes.

In case of protected inheritance, all the members of the base class with public and protected access become protected. It means that if public data members and methods of the base class were accessible from the outside, with protected inheritance they are available only from the classes of the derived class and its further derivatives.

```
//+------------------------------------------------------------------+
//| Example class with a few access types                            |
//+------------------------------------------------------------------+
class CBaseMathClass
  {
private:             //--- The private member is not available from derive
   double            m_Pi;
public:              //--- Getting and setting a value for m_Pi
   void              SetPI(double v){m_Pi=v;return;};
   double            GetPI(){return m_Pi;};
public:              // The class constructor is available to all members
                     CBaseMathClass() {SetPI(3.14);  PrintFormat("%s",__FU
  };
//+------------------------------------------------------------------+
//| A derived class, in which m_Pi cannot be modified                |
//+------------------------------------------------------------------+
class CProtectedChildClass: protected CBaseMathClass // Protected inherita
  {
private:
   double            m_radius;
public:              //--- Public methods in the derived class
   void              SetRadius(double r){m_radius=r; return;};
   double            GetCircleLength(){return GetPI()*m_radius;};
  };
//+------------------------------------------------------------------+
//| Script starting function                                         |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- When creating a derived class, the constructor of the base class wil
   CProtectedChildClass pt;
//--- Specify radius
   pt.SetRadius(10);
   PrintFormat("Length=%G",pt.GetCircleLength());
//--- If we uncomment the line below, we will get an error at the stage of
// pt.SetPI(3);

//--- Now declare a variable of the base class and try to set the Pi const
   CBaseMathClass bc;
   bc.SetPI(10);
//--- Here is the result
   PrintFormat("bc.GetPI()=%G",bc.GetPI());
  }
```

The example shows that methods SetPI() and GetPi() in the base class CBaseMathClass are open and available for calling from any place of the program. But at the same time, for CProtectedChildClass which is derived from it these methods can be called only from the methods of the CProtectedChildClass class or its derived classes.

In case of private inheritance, all the members of the basic class with the public and protected access become private, and calling them becomes impossible in further inheritance.

MQL4 has no multiple inheritance.

See also

Structures and Classes

Polymorphism

Polymorphism is an opportunity for different classes of objects, related through inheritance, to respond in various ways when calling the same function element. It helps to create a universal mechanism describing the behavior of not only the base class, but also descendant classes.

Let's continue to develop a base class CShape, and define a member function GetArea(), designed to calculate the area of a shape. In all the descendant classes, produced by inheritance from the base class, we redefine this function in accordance with rules of calculating the area of a particular shape.

For a square (class CSquare), the area is calculated through its sides, for a circle (class CCircle), area is expressed through its radius etc. We can create an array to store objects of CShape type, in which both objects of a base class and those of all descendant classes can be stored. Further we can call the same function for each element of the array.

Example:

```
//--- Base class class CShape
  {
protected:
   int            m_type;                // Shape type
   int            m_xpos;                // X - coordinate of the base poi
   int            m_ypos;                // Y - coordinate of the base poi
public:
   void           CShape(){m_type=0;};   // constructor, type=0
   int            GetType(){return(m_type);};// returns type of the shape
virtual
   double         GetArea(){return (0); }// returns area of the shape
  };
```

Now, all of the derived classes have a member function getArea(), which returns a zero value. The implementation of this function in each descendant will vary.

```
//--- The derived class Circle
class CCircle : public CShape            // After a colon we define the ba
  {                                      // from which inheritance is made
private:
   double         m_radius;              // circle radius

public:
   void           CCircle(){m_type=1;};  // constructor, type=1
   void           SetRadius(double r){m_radius=r;};
   virtual double GetArea(){return (3.14*m_radius*m_radius);}// circle are
  };
```

For the class Square the declaration is the same:

```
//--- The derived class Square
class CSquare : public CShape            // After a colon we define the ba
  {                                      // from which inheritance is made
private:
   double          m_square_side;        // square side

public:
   void            CSquare(){m_type=2;}; // constructor, type=1
   void            SetSide(double s){m_square_side=s;};
   virtual double  GetArea(){return (m_square_side*m_square_side);}// squa
  };
```

For calculating the area of the square and circle, we need the corresponding values of m_radius and m_square_side, so we have added the functions SetRadius() and SetSide() in the declaration of the corresponding class.

It is assumed that object of different types (CCircle and CSquare) derived from one base type CShape are used in our program. Polymorphism allows creating an array of objects of the base CShape class, but when declaring this array, these objects are yet unknown and their type is undefined.

The decision on what type of object will be contained in each element of the array will be taken directly during program execution. This involves the dynamic creation of objects of the appropriate classes, and hence the necessity to use object pointers instead of objects.

The new operator is used for dynamic creation of objects. Each such object must be individually and explicitly deleted using the delete operator. Therefore we will declare an array of pointers of CShape type, and create an object of a proper type for each element (**new Class_Name**), as shown in the following script example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Declare an array of object pointers of the base type
   CShape *shapes[5];   // An array of pointers to CShape object

//--- Here fill in the array with derived objects
//--- Declare a pointer to the object of CCircle type
   CCircle *circle=new CCircle();
//--- Set object properties at the circle pointer
   circle.SetRadius(2.5);
//--- Place the pointer value in shapes[0]
   shapes[0]=circle;

//--- Create another CCircle object and write down its pointer in shapes[1
   circle=new CCircle();
   shapes[1]=circle;
   circle.SetRadius(5);

//--- Here we intentionally "forget" to set a value for shapes[2]
//circle=new CCircle();
//circle.SetRadius(10);
//shapes[2]=circle;

//--- Set NULL for the element that is not used
   shapes[2]=NULL;

//--- Create a CSquare object and write down its pointer to shapes[3]
   CSquare *square=new CSquare();
   square.SetSide(5);
   shapes[3]=square;

//--- Create a CSquare object and write down its pointer to shapes[4]
   square=new CSquare();
   square.SetSide(10);
   shapes[4]=square;

//--- We have an array of pointers, get its size
   int total=ArraySize(shapes);
//--- Pass in a loop through all pointers in the array
   for(int i=0; i<5;i++)
     {
      //--- If the pointer at the specified index is valid
      if(CheckPointer(shapes[i])!=POINTER_INVALID)
        {
         //--- Log the type and square of the shape
         PrintFormat("The object of type %d has the square %G",
               shapes[i].GetType(),
               shapes[i].GetArea());
        }
      //--- If the pointer has type POINTER_INVALID
      else
        {
         //--- Notify of an error
         PrintFormat("Object shapes[%d] has not been initialized! Its poin
                     i,EnumToString(CheckPointer(shapes[i])));
        }
     }

//--- We must delete all created dynamic objects
   for(int i=0;i<total;i++)
     {
      //--- We can delete only the objects with pointers of POINTER_DYNAMI
      if(CheckPointer(shapes[i])==POINTER_DYNAMIC)
        {
         //--- Notify of deletion
         PrintFormat("Deleting shapes[%d]",i);
         //--- Delete an object by its pointer
         delete shapes[i];
        }
     }
  }
```

Please note that when deleting an object using the delete operator, the type of its pointer must be checked. Only objects with the POINTER_DYNAMIC pointer can be deleted using delete. For pointers of other type, an error will be returned.

But besides the redefining of functions during inheritance, polymorphism also includes the implementation of one and the same functions with different sets of parameters within a class. This means that the class may have several functions with the same name but with a different type and/or set of parameters. In this case, polymorphism is implemented through the function overload.

Overload

Within one class it is possible to define two or more methods that use the same name, but have different numbers of parameters. When this occurs, methods are called overloaded and such a process is referred to as method overloading.

Method overloading is one of ways of polymorphism realization. Overloading of methods is performed according to the same rules as the function overloading.

If the called function has no exact match, the compiler searches for a suitable function on three levels sequentially:

1. search within class methods.
2. search within the base class methods, consistently from the nearest ancestor to the very first.
3. search among other functions.

If there is no exact correspondence at all levels, but several suitable functions at different levels have been found, the function found at the least level is used. Within one level, there can't be more than one suitable function.

See also

Function Overloading

Virtual Functions

The virtual keyword is the function specifier, which provides a mechanism to select dynamically at runtime an appropriate function-member among the functions of basic and derived classes. Structures cannot have virtual functions. It can be used to change the declarations for function-members only.

The virtual function, like an ordinary function, must have an executable body. When called, its semantic is the same as that of other functions.

A virtual function may be overridden in a derived class. The choice of what function definition should be called for a virtual function is made dynamically (at runtime). A typical case is when a base class contains a virtual function, and derived classes have their own versions of this function.

The pointer to the base class can indicate either a base class object or the object of a derived class. The choice of the member-function to call will be performed at runtime and will depend on the type of the object, not the type of the pointer. If there is no member of a derived type, the virtual function of the base class is used by default.

Destructors are always virtual, regardless of whether they are declared with the virtual keyword or not.

Attention: it is not recommended to call virtual methods from constructors and desctructors, because the result is undefined in this case.

Let's consider the use of virtual functions on the example of Tetris.mq5. The base class CTetrisShape with the virtual function Draw is defined in the included file TetisShape.mqh.

```
//+-------------------------------------------------------------------+ cla
  {
protected:
   int                  m_type;
   int                  m_xpos;
   int                  m_ypos;
   int                  m_xsize;
   int                  m_ysize;
   int                  m_prev_turn;
   int                  m_turn;
   int                  m_right_border;
public:
   void                 CTetrisShape();
   void                 SetRightBorder(int border) { m_right_border=border; }
   void                 SetYPos(int ypos)          { m_ypos=ypos;           }
   void                 SetXPos(int xpos)          { m_xpos=xpos;           }
   int                  GetYPos()                  { return(m_ypos);        }
   int                  GetXPos()                  { return(m_xpos);        }
   int                  GetYSize()                 { return(m_ysize);       }
   int                  GetXSize()                 { return(m_xsize);       }
   int                  GetType()                  { return(m_type);        }
   void                 Left()                     { m_xpos-=SHAPE_SIZE;    }
   void                 Right()                    { m_xpos+=SHAPE_SIZE;    }
   void                 Rotate()                   { m_prev_turn=m_turn; if(+
   virtual void         Draw()                     { return;                }
   virtual bool         CheckDown(int& pad_array[]);
   virtual bool         CheckLeft(int& side_row[]);
   virtual bool         CheckRight(int& side_row[]);
  };
```

Further, for each derived class, this function is implemented in accordance with characteristics of a descendant class. For example, the first shape CTetrisShape1 has its own implementation of the Draw() function:

```
class CTetrisShape1 : public CTetrisShape
  {
public:
   //--- shape drawing
   virtual void      Draw()
     {
      int    i;
      string name;
      //---
      if(m_turn==0 || m_turn==2)
        {
         //--- horizontal
         for(i=0; i<4; i++)
           {
            name=SHAPE_NAME+(string)i;
            ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+i*SHAPE_SIZE)
            ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
           }
        }
      else
        {
         //--- vertical
         for(i=0; i<4; i++)
           {
            name=SHAPE_NAME+(string)i;
            ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos);
            ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+i*SHAPE_SIZE)
           }
        }
     }
  }
```

The Square shape is described by class CTetrisShape6 and has its own implementation of the Draw() method:

```
class CTetrisShape6 : public CTetrisShape
  {
public:
   //--- Shape drawing
   virtual void      Draw()
     {
      int    i;
      string name;
      //---
      for(i=0; i<2; i++)
        {
         name=SHAPE_NAME+(string)i;
         ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+i*SHAPE_SIZE);
         ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos);
        }
      for(i=2; i<4; i++)
        {
         name=SHAPE_NAME+(string)i;
         ObjectSetInteger(0,name,OBJPROP_XDISTANCE,m_xpos+(i-2)*SHAPE_SIZE
         ObjectSetInteger(0,name,OBJPROP_YDISTANCE,m_ypos+SHAPE_SIZE);
        }
     }
  };
```

Depending on the class, to which the created object belongs, it calls the virtual function of this or that derived class.

```
void CTetrisField::NewShape()
  {
//--- creating one of the 7 possible shapes randomly
   int nshape=rand()%7;
   switch(nshape)
     {
      case 0: m_shape=new CTetrisShape1; break;
      case 1: m_shape=new CTetrisShape2; break;
      case 2: m_shape=new CTetrisShape3; break;
      case 3: m_shape=new CTetrisShape4; break;
      case 4: m_shape=new CTetrisShape5; break;
      case 5: m_shape=new CTetrisShape6; break;
      case 6: m_shape=new CTetrisShape7; break;
     }
//--- draw
   m_shape.Draw();
//---
  }
```

Static members of a Class/Structure

Static Members

The members of a class can be declared using the storage class modifier static. These data members are shared by all instances of this class and are stored in one place. Non-static data members are created for each class object variable.

The inability to declare static members of a class would have led to the need to declare these data on the the global level of the program. It would break the relationship between the data and their class, and is not consistent with the basic paradigm of the OOP - joining data and methods for handling them in a class. The static member allows class data that are not specific to a particular instance to exist in the class scope.

Since a static class member does not depend on the particular instance, the reference to it is as follows:

```
class_name::variable
```

where *class_name* is the name of the class, and *variable* is the name of the class member.

As you see, to access the static member of a class, context resolution operator :: is used. When you access a static member within class methods, the context operator is optional.

Static member of a class has to be explicitly initialized with desired value. For this it must be declared and initialized in global scope. The sequence of static members initialization will correspond to the sequence of their declaration in global scope.

For example, we have a class *CParser* used for parsing the text, and we need to count the total number of processed words and characters. We only need to declare the necessary class members as static and initialize them at the global level. Then all instances of the class will use common counters of words and characters.

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
class CParser
  {
public:
   static int        s_words;
   static int        s_symbols;
   //--- Constructor and destructor
                     CParser(void);
                    ~CParser(void){};
  };
...
//--- Initialization of static members of the Parser class at the global l
int CParser::s_words=0;
int CParser::s_symbols=0;
```

A static class member can be declared with the *const* keyword. Such static constants must be initialized at the global level with the *const* keyword:

```
//+------------------------------------------------------------------+
//| Class "Stack" for storing processed data                         |
//+------------------------------------------------------------------+
class CStack
  {
public:
                     CStack(void);
                    ~CStack(void){};
...
private:
   static const int  s_max_length; // Maximum stack capacity
  };

//--- Initialization of the static constant of the CStack class
const int CStack::s_max_length=1000;
```

Pointer this

The keyword this denotes an implicitly declared pointer to itself to a specific instance of the class, in the context of which the method is executed. It can be used only in non-static methods of the class. Pointer this is an implicit non-static member of any class.

In static functions you can access only static members/methods of a class.

Static Methods

In MQL4 member functions of type static can be used. The *static* modifier must precede the return type of a function in the declaration inside a class.

```
class CStack
  {
public:
   //--- Constructor and destructor
                     CStack(void){};
                    ~CStack(void){};
   //--- Maximum stack capacity
   static int        Capacity();
private:
   int               m_length;     // The number of elements in the stack
   static const int  s_max_length; // Maximum stack capacity
  };
//+------------------------------------------------------------------+
//| Returns the maximum number of elements to store in the stack     |
//+------------------------------------------------------------------+
int CStack::Capacity(void)
  {
   return(s_max_length);
  }
//--- Initialization of the static constant of the CStack class
const int CStack::s_max_length=1000;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declare CStack type variable
   CStack stack;
//--- call the object's static method
   Print("CStack.s_max_length=",stack.Capacity());
//--- it can also be called the following way, as the method is static and
   Print("CStack.s_max_length=",CStack::Capacity());
  }
```

A method with the const modifier is called constant and cannot modify implicit members of its class. Declaration of constant functions of a class and constant parameters is called *const-correctness* control. Through this control you can be sure that the compiler will ensure the consistency of values of objects and will return an error during compilation if there is something wrong.

The const modifier is placed after the list of arguments inside a class declaration. Definition outside a class should also include the *const* modifier:

```
//+------------------------------------------------------------------+
//| Class "Rectangle"                                                |
//+------------------------------------------------------------------+
class CRectangle
  {
private:
   double            m_width;      // Width
   double            m_height;     // Height
public:
   //--- Constructors and destructor
                     CRectangle(void):m_width(0),m_height(0){};
                     CRectangle(const double w,const double h):m_width(w),
                    ~CRectangle(void){};
   //--- Calculating the area
   double            Square(void) const;
   static double     Square(const double w,const double h);// { return(w*h
  };
//+------------------------------------------------------------------+
//| Returns the area of the "Rectangle" object                       |
//+------------------------------------------------------------------+
double CRectangle::Square(void) const
  {
   return(Square(m_width,m_height));
  }
//+------------------------------------------------------------------+
//| Returns the product of two variables                             |
//+------------------------------------------------------------------+
static double CRectangle::Square(const double w,const double h)
  {
   return(w*h);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Create a rectangle rect with the sides equal to 5 and 6
   CRectangle rect(5,6);
//--- Find the rectangle area using a constant method
   PrintFormat("rect.Square()=%.2f",rect.Square());
//--- Find the product of numbers using the static method of class CRectar
   PrintFormat("CRectangle::Square(2.0,1.5)=%f",CRectangle::Square(2.0,1.5
  }
```

An additional argument in favor of using the constancy control is the fact that in this case, the compiler generates a special optimization, for example, places a constant object in read-only memory.

A static function cannot be determined with the const modifier, because this modifier ensures the constancy of the instance members when calling this function. But, as mentioned above, the static function cannot access non-static class members.

See also

Static Variables, Variables, References. Modifier & and Keyword this

Function Templates

Overloaded functions are commonly used to perform similar operations on various data types. ArraySize() is a simple example of such function in MQL4. It returns size of any type of array. In fact, this system function is overloaded and the entire implementation of such an overload is hidden from MQL4 application developers:

```
int  ArraySize(    void&  array[]      // checked array
   );
```

It means that MQL4 language compiler inserts necessary implementation for each call of this function. For example, that is how it can be done for integer type arrays:

```
int  ArraySize(
   int&  array[]      // array with int type elements
   );
```

ArraySize() function can be displayed the following way for MqlRates type array for working with quotations in historical data format:

```
int  ArraySize(
   MqlRates&  array[] // array filled with MqlRates type values
   );
```

Thus, it is very convenient to use the same function for working with different types. However, all preliminary work should be carried out the necessary function should be overloaded for all data types it should correctly work with.

There is a convenient solution. If similar operations should be executed for each data type, it is possible to use function templates. In this case, a programmer needs to write only one function template description. When describing the template in such a way, we should specify only some formal parameter instead of some definite data type the function should work with. The compiler will automatically generate various functions for the appropriate handling of each type based on the types of the arguments used when calling the function.

Function template definition starts with the template keyword followed by the list of formal parameters in angle brackets. Each formal parameter is preceded by the typename keyword. Formal parameter types are built-in or user-defined types. They are used:

· to specify the types of function arguments,
· to specify the types of function's return value,
· to declare the variables inside the function definition

Number of template parameters cannot exceed eight. Each formal parameter in the template definition should appear in the list of function parameters at least once. Each name of the formal parameter should be unique.

Below is an example of a function template for searching the highest value in the array of any numeric type (integer and real numbers):

```
template<typename T>
T ArrayMax(T &arr[])
  {
   uint size=ArraySize(arr);
   if(size==0) return(0);

   T max=arr[0];
   for(uint n=1;n<size;n++)
      if(max<arr[n]) max=arr[n];
//---
   return(max);
  }
```

This template defines the function that finds the highest value in the passed array and returns this value as a result. Keep in mind that the ArrayMaximum() function built in MQL4 returns only the highest value index that can be used to find the value itself. For example:

```
//--- create an array
   double array[];
   int size=50;
   ArrayResize(array,size);
//---  fill with random values
   for(int i=0;i<size;i++)
     {
      array[i]=MathRand();
     }

//--- find position of the highest value in the array
   int max_position=ArrayMaximum(array);
//--- now, get the highest value itself in the array
   double max=array[max_position];
//--- display the found value
   Print("Max value = ",max);
```

Thus, we have performed two steps to get the highest value in the array. With ArrayMax() function template, we can get the result of the necessary type just by passing the array of an appropriate type into this function. It means that instead of two last lines

```
//--- find position of the highest value in the array
   int max_position=ArrayMaximum(array);
//--- now, receive the highest value itself in the array
   double max=array[max_position];
```

we now can use only one line, in which the returned result has the same type as the array passed into function:

```
//--- find the highest value
   double max=ArrayMax(array);
```

In this case, the type of result returned by the ArrayMax() function will automatically match the type of array.

Use the typename keyword to get the argument type as a string in order to create general purpose methods of working with various data types. Let's consider a specific example of the function that returns data type as a string:

```
#include <Trade\Trade.mqh>
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   CTrade trade;
   double d_value=M_PI;
   int i_value=INT_MAX;
   Print("d_value: type=",GetTypeName(d_value), ",   value=", d_value);
   Print("i_value: type=",GetTypeName(i_value), ",   value=", i_value);
   Print("trade: type=",GetTypeName(trade));
//---
  }
//+------------------------------------------------------------------+
//| Type is returned as a line                                       |
//+------------------------------------------------------------------+
template<typename T>
string GetTypeName(const T &t)
  {
//--- return the type as a line
   return(typename(T));
//---
  }
```

Function templates can also be used for class methods, for example:

```
class CFile
  {
   ...
public:
   ...
   template<typename T>
   uint WriteStruct(T &data);
  };

template<typename T>
uint CFile::WriteStruct(T &data)
  {
   ...
   return(FileWriteStruct(m_handle,data));
  }
```

Function templates should not be declared with export, virtual and #import keywords.

Abstract Classes and Pure Virtual Functions

Abstract classes are used for creating generic entities, that you expect to use for creating more specific derived classes. An abstract class can only be used as the base class for some other class, that is why it is impossible to create an object of the abstract class type.

A class which contains at least one pure virtual function in it is abstract. Therefore, classes derived from the abstract class must implement all its pure virtual functions, otherwise they will also be abstract classes.

A virtual function is declared as "pure" by using the pure-specifier syntax. Consider the example of the CAnimal class, which is only created to provide common functions the objects of the CAnimal type are too general for practical use. Thus, CAnimal is a good example for an abstract class:

```
class CAnimal   {
public:
                      CAnimal();     // Constructor
   virtual void       Sound() = 0;   // A pure virtual function
private:
   double             m_legs_count;  // The number of the animal's legs
  };
```

Here Sound() is a pure virtual function, because it is declared with the specifier of the pure virtual function PURE (**=0**).

Pure virtual functions are only the virtual functions for which the PURE specifier is set: (=NULL) or (=0). Example of abstract class declaration and use:

```
class CAnimal
  {
public:
   virtual void          Sound()=NULL;   // PURE method, should be overridde
  };
//--- Derived from an abstract class
class CCat : public CAnimal
 {
public:
  virtual void          Sound() { Print("Myau"); } // PURE is overridden, CC
 };

//--- Examples of wrong use
new CAnimal;         // Error of 'CAnimal' - the compiler returns the "can
CAnimal some_animal; // Error of 'CAnimal' - the compiler returns the "can

//--- Examples of proper use
new CCat;  // No error - the CCat class is not abstract
CCat cat;  // No error - the CCat class is not abstract
```

Restrictions on abstract classes

If the constructor for an abstract class calls a pure virtual function (either directly or indirectly), the result is undefined.

```
//+------------------------------------------------------------------+
//| An abstract base class                                           |
//+------------------------------------------------------------------+
class CAnimal
  {
public:
   //--- A pure virtual function
   virtual void      Sound(void)=NULL;
   //--- Function
   void              CallSound(void) { Sound(); }
   //--- Constructor
   CAnimal()
    {
     //--- An explicit call of the virtual method
     Sound();
     //--- An implicit call (using a third function)
     CallSound();
     //--- A constructor and/or destructor always calls its own functions,
     //--- even if they are virtual and overridden by a called function in
     //--- If the called function is pure virtual,
     //--- its call will cause a critical runtime error: "pure virtual fun
    }
  };
```

However, constructors and destructors for abstract classes can call other member functions.

Standard Constants, Enumerations and Structures

To simplify the program writing and to make program texts more convenient for perception, the MQL4 language provides predefined standard constants and enumerations. Besides that, service structures are used for storing information.

Standard constants are similar to macros and are of int type.

The constants are grouped by their purposes:

- Chart constants are used when working with price charts: opening, navigation, setting parameters;
- Objects constants are intended for processing graphical objects that can be created and displayed in charts;
- Indicators constants are used for working with standard and custom indicators;
- Environment state constants describe properties of a MQL4-program, show information about a client terminal, financial instrument and current account;
- Trade constants allow to specify a variety of information in the course of trading;
- Named constants are constants of the MQL4 language;
- Data structures describe data storage formats used;
- Codes of errors and warnings describe compiler messages and trading server answers to trade requests;
- In/out constants are designed for working with file functions and displaying messages on the screen by the MessageBox() function.

Chart Constants

Constants describing various properties of charts are divided into the following groups:

- Types of events events that occur when working with charts;
- Chart timeframes standard built-in periods;
- Properties of chart identifiers that are used as parameters of chart functions;
- Positioning constants - value of a parameter of the ChartNavigate() function;
- Displaying charts - setting the chart appearance.

Types of Chart Events

There are 9 types of events that can be processed using the predefined function OnChartEvent(). For custom events 65535 identifiers are provided in the range of CHARTEVENT_CUSTOM to CHARTEVENT_CUSTOM_LAST inclusive. To generate a custom event, the EventChartCustom() function should be used.

ENUM_CHART_EVENT

ID	Description
CHARTEVENT_KEYDOWN	Keystrokes
CHARTEVENT_MOUSE_MOVE	Mouse move, mouse clicks (if CHART_EVENT_MOUSE_MOVE=true is set for the chart)
CHARTEVENT_OBJECT_CREATE	Graphical object created (if CHART_EVENT_OBJECT_CREATE=true is set for the chart)
CHARTEVENT_OBJECT_CHANGE	Graphical object property changed via the properties dialog
CHARTEVENT_OBJECT_DELETE	Graphical object deleted (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)
CHARTEVENT_CLICK	Clicking on a chart
CHARTEVENT_OBJECT_CLICK	Clicking on a graphical object
CHARTEVENT_OBJECT_DRAG	Drag and drop of a graphical object
CHARTEVENT_OBJECT_ENDEDIT	End of text editing in the graphical object Edit
CHARTEVENT_CHART_CHANGE	Change of the chart size or modification of chart properties through the Properties dialog
CHARTEVENT_CUSTOM	Initial number of an event from a range of custom events
CHARTEVENT_CUSTOM_LAST	The final number of an event from a range of custom events

For each type of event, the input parameters of the OnChartEvent() function have definite values that are required for the processing of this event. The events and values passed through this parameters are listed in the below table.

Event	Value of the id parameter	Value of the lpa

		parameter
Event of a keystroke	CHARTEVENT_KEYDOWN	code of a pressed key
Mouse events (if CHART_EVENT_MOUSE_MOVE=true is set for the chart)	CHARTEVENT_MOUSE_MOVE	the X coordinate
event of graphical object creation (if CHART_EVENT_OBJECT_CREATE=true is set for the chart)	CHARTEVENT_OBJECT_CREATE	
Event of change of an object property via the properties dialog	CHARTEVENT_OBJECT_CHANGE	
Event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)	CHARTEVENT_OBJECT_DELETE	
Event of a mouse click on the chart	CHARTEVENT_CLICK	the X coordinate
Event of a mouse click in a graphical object belonging to the chart	CHARTEVENT_OBJECT_CLICK	the X coordinate
Event of a graphical object dragging using the mouse	CHARTEVENT_OBJECT_DRAG	
Event of the finished text editing in the entry box of the LabelEdit graphical object	CHARTEVENT_OBJECT_ENDEDIT	
Event of change of the chart size or modification of chart properties through the Properties dialog	CHARTEVENT_CHART_CHANGE	
ID of the user event under the N number	CHARTEVENT_CUSTOM+N	Value set by the EventChartCusto

		function

Example:

```
#define KEY_NUMPAD_5       12 #define KEY_LEFT           37
#define KEY_UP             38
#define KEY_RIGHT          39
#define KEY_DOWN           40
#define KEY_NUMLOCK_DOWN   98
#define KEY_NUMLOCK_LEFT  100
#define KEY_NUMLOCK_5     101
#define KEY_NUMLOCK_RIGHT 102
#define KEY_NUMLOCK_UP    104
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   Print("The Expert Advisor with name ",MQLInfoString(MQL_PROGRAM_NAME),"
//--- enable object create events
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_CREATE,true);
//--- enable object delete events
   ChartSetInteger(ChartID(),CHART_EVENT_OBJECT_DELETE,true);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // Event identifier
                  const long& lparam,   // Event parameter of long type
                  const double& dparam, // Event parameter of double type
                  const string& sparam) // Event parameter of string type
  {
//--- the left mouse button has been pressed on the chart
   if(id==CHARTEVENT_CLICK)
     {
      Print("The coordinates of the mouse click on the chart are: x = ",lp
     }
//--- the mouse has been clicked on the graphic object
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      Print("The mouse has been clicked on the object with name '"+sparam+
     }
//--- the key has been pressed
   if(id==CHARTEVENT_KEYDOWN)
```

```
     {
      switch(int(lparam))
        {
         case KEY_NUMLOCK_LEFT:  Print("The KEY_NUMLOCK_LEFT has been pres
         case KEY_LEFT:          Print("The KEY_LEFT has been pressed");
         case KEY_NUMLOCK_UP:    Print("The KEY_NUMLOCK_UP has been presse
         case KEY_UP:            Print("The KEY_UP has been pressed");
         case KEY_NUMLOCK_RIGHT: Print("The KEY_NUMLOCK_RIGHT has been pre
         case KEY_RIGHT:         Print("The KEY_RIGHT has been pressed");
         case KEY_NUMLOCK_DOWN:  Print("The KEY_NUMLOCK_DOWN has been pres
         case KEY_DOWN:          Print("The KEY_DOWN has been pressed");
         case KEY_NUMPAD_5:      Print("The KEY_NUMPAD_5 has been pressed"
         case KEY_NUMLOCK_5:     Print("The KEY_NUMLOCK_5 has been pressed
         default:                Print("Some not listed key has been press
        }
      ChartRedraw();
     }
//--- the object has been deleted
   if(id==CHARTEVENT_OBJECT_DELETE)
     {
      Print("The object with name ",sparam," has been deleted");
     }
//--- the object has been created
   if(id==CHARTEVENT_OBJECT_CREATE)
     {
      Print("The object with name ",sparam," has been created");
     }
//--- the object has been moved or its anchor point coordinates has been c
   if(id==CHARTEVENT_OBJECT_DRAG)
     {
      Print("The anchor point coordinates of the object with name ",sparam
     }
//--- the text in the Edit of object has been changed
   if(id==CHARTEVENT_OBJECT_ENDEDIT)
     {
      Print("The text in the Edit field of the object with name ",sparam,"
     }
  }
```

For CHARTEVENT_MOUSE_MOVE event the **sparam** string parameter contains information about state of the keyboard and mouse buttons:

Bit	Description
1	State of the left mouse button
2	State of the right mouse button
3	State of the SHIFT button

4	State of the CTRL button
5	State of the middle mouse button
6	State of the first extra mouse button
7	State of the second extra mouse button

Example:

```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- enable CHART_EVENT_MOUSE_MOVE messages
   ChartSetInteger(0,CHART_EVENT_MOUSE_MOVE,1);
  }
//+------------------------------------------------------------------+
//| MouseState                                                       |
//+------------------------------------------------------------------+
string MouseState(uint state)
  {
   string res;
   res+="\nML: "   +(((state& 1)== 1)?"DN":"UP");   // mouse left
   res+="\nMR: "   +(((state& 2)== 2)?"DN":"UP");   // mouse right
   res+="\nMM: "   +(((state&16)==16)?"DN":"UP");   // mouse middle
   res+="\nMX: "   +(((state&32)==32)?"DN":"UP");   // mouse first X key
   res+="\nMY: "   +(((state&64)==64)?"DN":"UP");   // mouse second X key
   res+="\nSHIFT: "+(((state& 4)== 4)?"DN":"UP");   // shift key
   res+="\nCTRL: " +(((state& 8)== 8)?"DN":"UP");   // control key
   return(res);
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,const long &lparam,const double &dparam,cor
  {
   if(id==CHARTEVENT_MOUSE_MOVE)
      Comment("POINT: ",(int)lparam,",",(int)dparam,"\n",MouseState((uint)
  }
```

See also

[Event Handling Functions](), [Working with events]()

Chart Timeframes

All predefined timeframes of charts have unique identifiers. The PERIOD_CURRENT identifier means the current period of a chart, at which a mql4-program is running.

ENUM_TIMEFRAMES

ID	Value	Description
PERIOD_CURRENT	0	Current timeframe
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 hours
PERIOD_D1	1440	1 day
PERIOD_W1	10080	1 week
PERIOD_MN1	43200	1 month

The ENUM_TIMEFRAMES enumeration contains the values of standard timeframes, online charts of financial instruments can be plotted only on these time intervals.

Below are non-standard timeframes, in MQL4 they are constants.

Constant	Value	Description
PERIOD_M2	2	2 minutes
PERIOD_M3	3	3 minutes
PERIOD_M4	4	4 minutes
PERIOD_M6	6	6 minutes
PERIOD_M10	10	10 minutes
PERIOD_M12	12	12 minutes
PERIOD_M20	20	20 minutes
PERIOD_H2	120	2 hours

PERIOD_H3	180	3 hours
PERIOD_H6	360	6 hours
PERIOD_H8	480	8 hours
PERIOD_H12	720	12 hours

These periods can be used for working with offline charts.

Note

The constants of non-standard timeframes are included in the MQL4 language to enable translation and compilation of MQL5 programs, where these timeframes are standard and are included in the ENUM_TIMEFRAMES enumeration.

See also

PeriodSeconds(), Period(), Date and Time, Visibility of objects

Chart Properties

Identifiers of ENUM_CHART_PROPERTY enumerations are used as parameters of functions for working with charts. The abbreviation of r/o in the "Property Type" column means that this property is read-only and cannot be changed. The w/o abbreviation in the "Property Type" column means that this property is write-only and it cannot be received. When accessing certain properties, it's necessary to specify an additional parameter-modifier (modifier), which serves to indicate the number of chart subwindows. 0 means the main window.

For functions ChartSetInteger() and ChartGetInteger()

ENUM_CHART_PROPERTY_INTEGER

ID	Description	Property Type
CHART_BRING_TO_TOP	Show chart on top of other charts	bool w/o
CHART_MOUSE_SCROLL	Scrolling the chart horizontally using the left mouse button. Vertical scrolling is also available if the value of any following properties is set to true: CHART_SCALEFIX, CHART_SCALEFIX_11 or CHART_SCALE_PT_PER_BAR	bool
CHART_EVENT_MOUSE_MOVE	Send notifications of mouse move and mouse click events (CHARTEVENT_MOUSE_MOVE) to all mql4 programs on a chart	bool
CHART_EVENT_OBJECT_CREATE	Send a notification of an event of new object creation (CHARTEVENT_OBJECT_CREATE) to all mql4-programs on a chart	bool
CHART_EVENT_OBJECT_DELETE	Send a notification of an event of object deletion (CHARTEVENT_OBJECT_DELETE) to all mql4-programs on a chart	bool
CHART_MODE	Chart type (candlesticks, bars or line)	enum ENUM_CHART

CHART_FOREGROUND	Price chart in the foreground	bool
CHART_SHIFT	Mode of price chart indent from the right border	bool
CHART_AUTOSCROLL	Mode of automatic moving to the right border of the chart	bool
CHART_SCALE	Scale	int from 0 to 5
CHART_SCALEFIX	Fixed scale mode	bool
CHART_SCALEFIX_11	Scale 1:1 mode	bool
CHART_SCALE_PT_PER_BAR	Scale to be specified in points per bar	bool
CHART_SHOW_OHLC	Show OHLC values in the upper left corner	bool
CHART_SHOW_BID_LINE	Display Bid values as a horizontal line in a chart	bool
CHART_SHOW_ASK_LINE	Display Ask values as a horizontal line in a chart	bool
CHART_SHOW_LAST_LINE	Display Last values as a horizontal line in a chart	bool
CHART_SHOW_PERIOD_SEP	Display vertical separators between adjacent periods	bool
CHART_SHOW_GRID	Display grid in the chart	bool
CHART_SHOW_VOLUMES	Display volume in the chart	enum ENUM_CHART_VOLUME
CHART_SHOW_OBJECT_DESCR	Display textual descriptions of objects (not available for all objects)	bool
CHART_VISIBLE_BARS	The number of bars on the chart that can be displayed	int r/o
CHART_WINDOWS_TOTAL	The total number of chart windows, including indicator subwindows	int r/o
CHART_WINDOW_IS_VISIBLE	Visibility of subwindows	bool r/o modifier - subwindow number
CHART_WINDOW_HANDLE	Chart window handle (HWND)	int r/o
CHART_WINDOW_YDISTANCE	The distance between the upper frame of the indicator	int r/o modifier - subwindow number

	subwindow and the upper frame of the main chart window, along the vertical Y axis, in pixels. In case of a mouse event, the cursor coordinates are passed in terms of the coordinates of the main chart window, while the coordinates of graphical objects in an indicator subwindow are set relative to the upper left corner of the subwindow. The value is required for converting the absolute coordinates of the main chart to the local coordinates of a subwindow for correct work with the graphical objects, whose coordinates are set relative to the upper left corner of the subwindow frame.	
CHART_FIRST_VISIBLE_BAR	Number of the first visible bar in the chart. Indexing of bars is the same as for timeseries.	int r/o
CHART_WIDTH_IN_BARS	Chart width in bars	int r/o
CHART_WIDTH_IN_PIXELS	Chart width in pixels	int r/o
CHART_HEIGHT_IN_PIXELS	Chart height in pixels	int modifier - subw number
CHART_COLOR_BACKGROUND	Chart background color	color
CHART_COLOR_FOREGROUND	Color of axes, scales and OHLC line	color
CHART_COLOR_GRID	Grid color	color
CHART_COLOR_VOLUME	Color of volumes and order opening levels	color
CHART_COLOR_CHART_UP	Color for the up bar, shadows and body borders of bull candlesticks	color
CHART_COLOR_CHART_DOWN	Color for the down bar,	color

	shadows and body borders of bear candlesticks	
CHART_COLOR_CHART_LINE	Line chart color and color of "Doji" Japanese candlesticks	color
CHART_COLOR_CANDLE_BULL	Body color of a bull candlestick	color
CHART_COLOR_CANDLE_BEAR	Body color of a bear candlestick	color
CHART_COLOR_BID	Bid price level color	color
CHART_COLOR_ASK	Ask price level color	color
CHART_COLOR_LAST	Line color of the last executed deal price (Last)	color
CHART_COLOR_STOP_LEVEL	Color of stop order levels (Stop Loss and Take Profit)	color
CHART_SHOW_TRADE_LEVELS	Displaying trade levels in the chart (levels of open orders, Stop Loss, Take Profit and pending orders)	bool
CHART_DRAG_TRADE_LEVELS	Permission to drag trading levels on a chart with a mouse. The drag mode is enabled by default (true value)	bool
CHART_SHOW_DATE_SCALE	Showing the time scale on a chart	bool
CHART_SHOW_PRICE_SCALE	Showing the price scale on a chart	bool
CHART_IS_OFFLINE	Flag, indicating that chart opened in offline mode	bool r/o

For functions ChartSetDouble() and ChartGetDouble()

ENUM_CHART_PROPERTY_DOUBLE

ID	Description	Property Type
CHART_SHIFT_SIZE	The size of the zero bar indent from the right border in percents	double (from 10 to 50 percents)
CHART_FIXED_POSITION	Chart fixed position from the left border in percent value. Chart fixed position is marked by a small gray triangle on the	double

	horizontal time axis. It is displayed only if the automatic chart scrolling to the right on tick incoming is disabled (see CHART_AUTOSCROLL property). The bar on a fixed position remains in the same place when zooming in and out.	
CHART_FIXED_MAX	Fixed chart maximum	double
CHART_FIXED_MIN	Fixed chart minimum	double
CHART_POINTS_PER_BAR	Scale in points per bar	double
CHART_PRICE_MIN	Chart minimum	double r/o modifier - subwindow number
CHART_PRICE_MAX	Chart maximum	double r/o modifier - subwindow number

For functions ChartSetString() and ChartGetString()

ENUM_CHART_PROPERTY_STRING

ID	Description	Property Type
CHART_COMMENT	Text of a comment in a chart	string

Example:

```
   int chartMode=ChartGetInteger(0,CHART_MODE);    switch(chartMode)
     {
      case(CHART_BARS):    Print("CHART_BARS");   break;
      case(CHART_CANDLES): Print("CHART_CANDLES");break;
      default:Print("CHART_LINE");
     }
   bool shifted=ChartGetInteger(0,CHART_SHIFT);
   if(shifted) Print("CHART_SHIFT = true");
   else Print("CHART_SHIFT = false");
   bool autoscroll=ChartGetInteger(0,CHART_AUTOSCROLL);
   if(autoscroll) Print("CHART_AUTOSCROLL = true");
   else Print("CHART_AUTOSCROLL = false");
   int chartHandle=ChartGetInteger(0,CHART_WINDOW_HANDLE);
   Print("CHART_WINDOW_HANDLE = ",chartHandle);
   int windows=ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   Print("CHART_WINDOWS_TOTAL = ",windows);
   if(windows>1)
     {
      for(int i=0;i<windows;i++)
        {
         int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,i);
         double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,i);
         double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,i);
         Print(i+": CHART_HEIGHT_IN_PIXELS = ",height," pixels");
         Print(i+": CHART_PRICE_MIN = ",priceMin);
         Print(i+": CHART_PRICE_MAX = ",priceMax);
        }
     }
```

See also

Examples of Working with the Chart

Positioning Constants

Three identifiers from the ENUM_CHART_POSITION list are the possible values of the *position* parameter for the ChartNavigate() function.

ENUM_CHART_POSITION

ID	Description
CHART_BEGIN	Chart beginning (the oldest prices)
CHART_CURRENT_POS	Current position
CHART_END	Chart end (the latest prices)

Example:

```
   long handle=ChartOpen("EURUSD",PERIOD_H12);    if(handle!=0)
     {
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      ChartSetInteger(handle,CHART_SHIFT,true);
      ChartSetInteger(handle,CHART_MODE,CHART_LINE);
      ResetLastError();
      bool res=ChartNavigate(handle,CHART_END,150);
      if(!res) Print("Navigate failed. Error = ",GetLastError());
      ChartRedraw();
     }
```

Chart Representation

Price charts can be displayed in three ways:

· as bars;

· as candlesticks;

· as a line.

The specific way of displaying the price chart is set by the function ChartSetInteger(chart_handle,CHART_MODE, chart_mode), where chart_mode is one of the values of the ENUM_CHART_MODE enumeration.

ENUM_CHART_MODE

ID	Description
CHART_BARS	Display as a sequence of bars
CHART_CANDLES	Display as Japanese candlesticks
CHART_LINE	Display as a line drawn by Close prices

To specify the mode of displaying volumes in the price chart the function ChartSetInteger(chart_handle, CHART_SHOW_VOLUMES, volume_mode) is used, where volume_mode is one of values of the ENUM_CHART_VOLUME_MODE enumeration.

ENUM_CHART_VOLUME_MODE

ID	Description
CHART_VOLUME_HIDE	Volumes are not shown
CHART_VOLUME_TICK	Tick volumes

Example:

```
//--- Get the handle of the current chart    long handle=ChartID();
   if(handle>0) // If it succeeded, additionally customize
     {
      //--- Disable autoscroll
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      //--- Set the indent of the right border of the chart
      ChartSetInteger(handle,CHART_SHIFT,true);
      //--- Display as candlesticks
      ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
      //--- Scroll by 100 bars from the beginning of history
      ChartNavigate(handle,CHART_CURRENT_POS,100);
      //--- Set the tick volume display mode
      ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);
     }
```

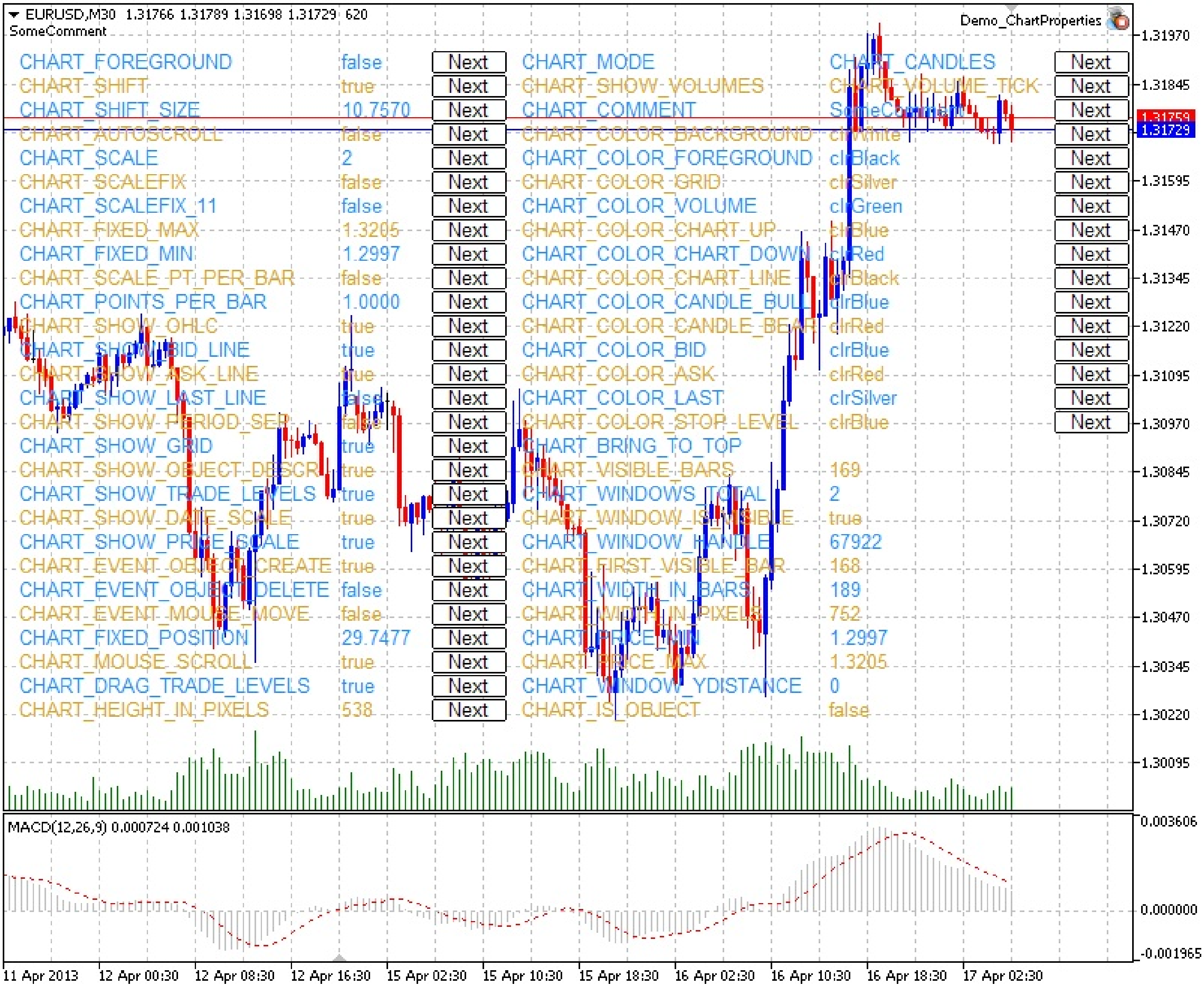
See also

ChartOpen, ChartID

Examples of Working with the Chart

This section contains examples of working with chart properties. One or two complete functions are displayed for each property. These functions allow setting/receiving the value of the property. These functions can be used "as is" in custom mql4 applications.

The screenshot below demonstrates the graphic panel illustrating how changing of the chart property changes its appearance. Clicking Next button allows setting the new value of the appropriate property and view the changes in the chart window.



EURUSD,M30 1.31766 1.31789 1.31698 1.31729 620
SomeComment

Demo_ChartProperties

CHART_FOREGROUND	false	Next	CHART_MODE	CHART_CANDLES	Next
CHART_SHIFT	true	Next	CHART_SHOW_VOLUMES	CHART_VOLUME_TICK	Next
CHART_SHIFT_SIZE	10.7570	Next	CHART_COMMENT	SomeComment	Next
CHART_AUTOSCROLL	false	Next	CHART_COLOR_BACKGROUND	clrWhite	Next
CHART_SCALE	2	Next	CHART_COLOR_FOREGROUND	clrBlack	Next
CHART_SCALEFIX	false	Next	CHART_COLOR_GRID	clrSilver	Next
CHART_SCALEFIX_11	false	Next	CHART_COLOR_VOLUME	clrGreen	Next
CHART_FIXED_MAX	1.3205	Next	CHART_COLOR_CHART_UP	clrBlue	Next
CHART_FIXED_MIN	1.2997	Next	CHART_COLOR_CHART_DOWN	clrRed	Next
CHART_SCALE_PT_PER_BAR	false	Next	CHART_COLOR_CHART_LINE	clrBlack	Next
CHART_POINTS_PER_BAR	1.0000	Next	CHART_COLOR_CANDLE_BULL	clrBlue	Next
CHART_SHOW_OHLC	true	Next	CHART_COLOR_CANDLE_BEAR	clrRed	Next
CHART_SHOW_BID_LINE	true	Next	CHART_COLOR_BID	clrBlue	Next
CHART_SHOW_ASK_LINE	true	Next	CHART_COLOR_ASK	clrRed	Next
CHART_SHOW_LAST_LINE	false	Next	CHART_COLOR_LAST	clrSilver	Next
CHART_SHOW_PERIOD_SEP	false	Next	CHART_COLOR_STOP_LEVEL	clrBlue	Next
CHART_SHOW_GRID	true	Next	CHART_BRING_TO_TOP		
CHART_SHOW_OBJECT_DESCR	true	Next	CHART_VISIBLE_BARS	169	
CHART_SHOW_TRADE_LEVELS	true	Next	CHART_WINDOWS_TOTAL	2	
CHART_SHOW_DATE_SCALE	true	Next	CHART_WINDOW_IS_VISIBLE	true	
CHART_SHOW_PRICE_SCALE	true	Next	CHART_WINDOW_HANDLE	67922	
CHART_EVENT_OBJECT_CREATE	true	Next	CHART_FIRST_VISIBLE_BAR	168	
CHART_EVENT_OBJECT_DELETE	false	Next	CHART_WIDTH_IN_BARS	189	
CHART_EVENT_MOUSE_MOVE	false	Next	CHART_WIDTH_IN_PIXELS	752	
CHART_FIXED_POSITION	29.7477	Next	CHART_PRICE_MIN	1.2997	
CHART_MOUSE_SCROLL	true	Next	CHART_PRICE_MAX	1.3205	
CHART_DRAG_TRADE_LEVELS	true	Next	CHART_WINDOW_YDISTANCE	0	
CHART_HEIGHT_IN_PIXELS	538	Next	CHART_IS_OBJECT	false	

MACD(12,26,9) 0.000724 0.001038

The panel's source code is located below.

Chart Properties and Sample Functions for Working with Them

- **CHART_BRING_TO_TOP** shows the chart on top of all others.

```
//+------------------------------------------------------------------+
//+------------------------------------------------------------------+
bool ChartBringToTop(const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- show the chart on top of all others
   if(!ChartSetInteger(chart_ID,CHART_BRING_TO_TOP,0,true))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_MOUSE_SCROLL** is a property for scrolling the chart using left mouse button.

```
//+-----------------------------------------------------------------------
//| The function defines if scrolling the chart using left mouse button is
//| enabled.
//+-----------------------------------------------------------------------
bool ChartMouseScrollGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------+
//| The function enables/disables scrolling the chart using left mouse |
//| button.                                                            |
//+--------------------------------------------------------------------+
bool ChartMouseScrollSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

· **CHART_EVENT_MOUSE_MOVE** is a property of sending messages concerning move events and mouse clicks to mql4 applications (CHARTEVENT_MOUSE_MOVE).

```
//+------------------------------------------------------------------+
//| Check if messages concerning move events and mouse clicks        |
//| are sent to all mql4 applications on the chart.                  |
//+------------------------------------------------------------------+
bool ChartEventMouseMoveGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+----------------------------------------------------------------------
//| The function enables/disables the mode of sending messages concerning
//| events and mouse clicks to mql4 applications on the
//| chart.
//+----------------------------------------------------------------------
bool ChartEventMouseMoveSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_EVENT_OBJECT_CREATE** is a property of sending messages concerning the event of a graphic object creation to mql4 applications (CHARTEVENT_OBJECT_CREATE).

```
//+------------------------------------------------------------------+
//| Check if messages concerning the event of a graphic object creation |
//| are sent to all mql4 applications on the chart.                  |
//+------------------------------------------------------------------+
bool ChartEventObjectCreateGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+-------------------------------------------------------------------
//| The function enables/disables the mode of sending messages concerning
//| the event of a graphic object creation to all mql4 applications on the
//| chart.
//+-------------------------------------------------------------------
bool ChartEventObjectCreateSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_EVENT_OBJECT_DELETE** is a property of sending messages concerning the event of a graphic object deletion to mql4 applications (CHARTEVENT_OBJECT_DELETE).

```
//+------------------------------------------------------------------+
//| Check if messages concerning the event of a graphic object deletion |
//| are sent to all mql4 applications on the chart.                  |
//+------------------------------------------------------------------+
bool ChartEventObjectDeleteGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------
//| The function enables/disables the mode of sending messages concerning
//| the event of a graphic object deletion to all mql4 applications on the
//| chart.
//+--------------------------------------------------------------------
bool ChartEventObjectDeleteSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_MODE** type of the chart (candlesticks, bars or line).

```
//+------------------------------------------------------------------+
//| Get chart display type (candlesticks, bars or                    |
//| line).                                                           |
//+------------------------------------------------------------------+
ENUM_CHART_MODE ChartModeGet(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=WRONG_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_MODE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((ENUM_CHART_MODE)result);
  }
//+------------------------------------------------------------------+
//| Set chart display type (candlesticks, bars or                    |
//| line).                                                           |
//+------------------------------------------------------------------+
bool ChartModeSet(const long value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_MODE,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_FOREGROUND** is a property of displaying a price chart in the foreground.

```
//+------------------------------------------------------------------+
//| The function defines if a price chart is displayed in the        |
//| foreground.                                                      |
//+------------------------------------------------------------------+
bool ChartForegroundGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_FOREGROUND,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+-----------------------------------------------------------------------
//| The function enables/disables the mode of displaying a price chart on
//| foreground.
//+-----------------------------------------------------------------------
bool ChartForegroundSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_FOREGROUND,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHIFT** mode of shift of the price chart from the right border.

```
//+----------------------------------------------------------------------
//| The function defines if the mode of shift of the price chart from the
//| is enabled.
//+----------------------------------------------------------------------
bool ChartShiftGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHIFT,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+----------------------------------------------------------------------
//| The function enables/disables the mode of displaying a price chart wit
//| a shift from the right border.
//+----------------------------------------------------------------------
bool ChartShiftSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHIFT,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_AUTOSCROLL** the mode of automatic shift to the right border of the chart.

```
//+---------------------------------------------------------------------+
//| The function defines if the mode of the autoscroll                  |
//| of the chart to the right in case of new ticks' arrival is enabled. |
//+---------------------------------------------------------------------+
bool ChartAutoscrollGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function enables/disables the mode of the autoscroll         |
//| of the chart to the right in case of new ticks' arrival.         |
//+------------------------------------------------------------------+
bool ChartAutoscrollSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SCALE** chart scale property.

```
//+------------------------------------------------------------------+
//| Get chart scale (from 0 to 5).                                   |
//+------------------------------------------------------------------+
int ChartScaleGet(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SCALE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
//+------------------------------------------------------------------+
//| Set chart scale (from 0 to 5).                                   |
//+------------------------------------------------------------------+
bool ChartScaleSet(const long value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

· **CHART_SCALEFIX** the mode of fixed chart scale.

```
//+------------------------------------------------------------------+
//| The function defines if the fixed scale mode is enabled.         |
//+------------------------------------------------------------------+
bool ChartScaleFixGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SCALEFIX,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function enables/disables the fixed scale mode.              |
//+------------------------------------------------------------------+
bool ChartScaleFixSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SCALEFIX,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SCALEFIX_11** 1:1 chart scale mode.

```
//+------------------------------------------------------------------+
//| The function defines if "1:1" scale is enabled.                  |
//+------------------------------------------------------------------+
bool ChartScaleFix11Get(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function enables/disables "1:1" scale mode                   |
//+------------------------------------------------------------------+
bool ChartScaleFix11Set(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SCALE_PT_PER_BAR** the mode of specifying the chart scale in points per bar.

```
//+------------------------------------------------------------------
//| The function defines if the mode of specifying the chart scale in poir
//| bar is enabled.
//+------------------------------------------------------------------
bool ChartScalePerBarGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------
//| The function enables/disables the mode of specifying the chart scale i
//| bar.
//+------------------------------------------------------------------
bool ChartScalePerBarSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_OHLC** the property of displaying OHLC values in the upper left corner.

```
//+------------------------------------------------------------------+
//| The function defines if the mode of displaying OHLC values       |
//| in the upper left corner is enabled.                             |
//+------------------------------------------------------------------+
bool ChartShowOHLCGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+-----------------------------------------------------------------------
//| The function enables/disables the mode of displaying OHLC values in th
//| upper left corner of the chart.
//+-----------------------------------------------------------------------
bool ChartShowOHLCSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_BID_LINE** the property of displaying Bid value as a horizontal line on the chart.

```
//+------------------------------------------------------------------------
//| The function defines if the mode of displaying Bid value line on the c
//| is enabled.
//+------------------------------------------------------------------------
bool ChartShowBidLineGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+----------------------------------------------------------------------+
//| The function enables/disables the mode of displaying Bid line on a |
//| chart.                                                               |
//+----------------------------------------------------------------------+
bool ChartShowBidLineSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_ASK_LINE** the property of displaying Ask value as a horizontal line on a chart.

```
//+------------------------------------------------------------------
//| The function defines if the mode of displaying Ask value line on the
//| chart.
//+------------------------------------------------------------------
bool ChartShowAskLineGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------
//| The function enables/disables the mode of displaying Ask line on the
//| chart.
//+------------------------------------------------------------------
bool ChartShowAskLineSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_LAST_LINE** the property of displaying Last value as a horizontal line on a chart.

```
//+------------------------------------------------------------------------
//| The function defines if the mode of displaying the line for the last p
//| deal's price is enabled.
//+------------------------------------------------------------------------
bool ChartShowLastLineGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------------
//| The function enables/disables the mode of displaying the line for the
//| deal's price.
//+------------------------------------------------------------------------
bool ChartShowLastLineSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_PERIOD_SEP** the property of displaying vertical separators between adjacent periods.

```
//+------------------------------------------------------------------+
//| The function defines if the mode of displaying vertical          |
//| separators between adjacent periods is enabled.                  |
//+------------------------------------------------------------------+
bool ChartShowPeriodSeparatorGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function enables/disables the mode of displaying vertical    |
//| separators between adjacent periods.                             |
//+------------------------------------------------------------------+
bool ChartShowPeriodSepapatorSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_GRID** the property of displaying the chart grid.

```
//+------------------------------------------------------------------+
//| The function defines if the chart grid is displayed.             |
//+------------------------------------------------------------------+
bool ChartShowGridGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_GRID,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function enables/disables the chart grid.                    |
//+------------------------------------------------------------------+
bool ChartShowGridSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_GRID,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_VOLUMES** the property of displaying the volumes on a chart.

```
//+------------------------------------------------------------------------
//| The function defines if the volumes are displayed on a chart (are not
//| displayed, tick ones are displayed, actual ones are displayed).
//+------------------------------------------------------------------------
ENUM_CHART_VOLUME_MODE ChartShowVolumesGet(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=WRONG_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_VOLUMES,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((ENUM_CHART_VOLUME_MODE)result);
  }
//+------------------------------------------------------------------+
//| The function sets the mode of displaying the volumes on a chart. |
//+------------------------------------------------------------------+
bool ChartShowVolumesSet(const long value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_VOLUMES,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_OBJECT_DESCR** the property of graphical object pop-up descriptions.

```
//+--------------------------------------------------------------------+
//| The function defines if pop-up descriptions                        |
//| of graphical objects are displayed when hovering mouse over them. |
//+--------------------------------------------------------------------+
bool ChartShowObjectDescriptionGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------------
//| The function enables/disables the mode of displaying pop-up descriptio
//| of graphical objects when hovering mouse over them.
//+--------------------------------------------------------------------------
bool ChartShowObjectDescriptionSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_VISIBLE_BARS** defines the number of bars on a chart that are available for display.

```
//+------------------------------------------------------------------------
//| The function receives the number of bars that are displayed (visible)
//| in the chart window.
//+------------------------------------------------------------------------
int ChartVisibleBars(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_VISIBLE_BARS,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WINDOWS_TOTAL** defines the total number of chart windows including indicator subwindows.

```
//+------------------------------------------------------------------------
//| The function gets the total number of chart windows including indicatc
//| subwindows.
//+------------------------------------------------------------------------
int ChartWindowsTotal(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WINDOWS_TOTAL,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WINDOW_IS_VISIBLE** defines the subwindow's visibility.

```
//+------------------------------------------------------------------+
//| The function defines if the current chart window or subwindow    |
//| is visible.                                                      |
//+------------------------------------------------------------------+
bool ChartWindowsIsVisible(bool &result,const long chart_ID=0,const int su
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_IS_VISIBLE,sub_window,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
```

· **CHART_WINDOW_HANDLE** returns the chart handle.

```
//+------------------------------------------------------------------+
//| The function gets the chart handle                               |
//+------------------------------------------------------------------+
int ChartWindowsHandle(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_HANDLE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WINDOW_YDISTANCE** defines the distance in pixels between the upper frame of the indicator subwindow and the upper frame of the chart's main window.

```
//+------------------------------------------------------------------+
//| The function gets the distance in pixels between the upper frame |
//| of the subwindow and the upper frame of the chart's main window. |
//+------------------------------------------------------------------+
int ChartWindowsYDistance(const long chart_ID=0,const int sub_window=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_YDISTANCE,sub_window,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_FIRST_VISIBLE_BAR** returns the number of the first visible bar on the chart (bar indexing corresponds to the time series).

```
//+---------------------------------------------------------------------------
//| The function receives the number of the first visible bar on the chart
//| Indexing is performed like in time series, last bars have smaller indi
//+---------------------------------------------------------------------------
int ChartFirstVisibleBar(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_YDISTANCE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WIDTH_IN_BARS** returns the chart width in bars.

```
//+------------------------------------------------------------------+
//| The function receives the chart width in bars.                   |
//+------------------------------------------------------------------+
int ChartWidthInBars(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_BARS,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_WIDTH_IN_PIXELS** returns the chart width in pixels.

```
//+------------------------------------------------------------------+
//| The function receives the chart width in pixels.                 |
//+------------------------------------------------------------------+
int ChartWidthInPixels(const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_PIXELS,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
```

- **CHART_HEIGHT_IN_PIXELS** chart height property in pixels.

```
//+------------------------------------------------------------------+
//| The function receives the chart height value in pixels.          |
//+------------------------------------------------------------------+
int ChartHeightInPixelsGet(const long chart_ID=0,const int sub_window=0)
  {
//--- prepare the variable to get the property value
   long result=-1;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((int)result);
  }
//+------------------------------------------------------------------+
//| The function sets the chart height value in pixels.              |
//+------------------------------------------------------------------+
bool ChartHeightInPixelsSet(const int value,const long chart_ID=0,const in
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

· **CHART_COLOR_BACKGROUND -** chart background color.

```
//+------------------------------------------------------------------+
//| The function receives chart background color.                    |
//+------------------------------------------------------------------+
color ChartBackColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive chart background color
   if(!ChartGetInteger(chart_ID,CHART_COLOR_BACKGROUND,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets chart background color.                        |
//+------------------------------------------------------------------+
bool ChartBackColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the chart background color
   if(!ChartSetInteger(chart_ID,CHART_COLOR_BACKGROUND,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_FOREGROUND** color of axes, scale and OHLC line.

```
//+------------------------------------------------------------------+
//| The function receives the color of axes, scale and OHLC line.   |
//+------------------------------------------------------------------+
color ChartForeColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of axes, scale and OHLC line
   if(!ChartGetInteger(chart_ID,CHART_COLOR_FOREGROUND,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets the color of axes, scale and OHLC line.       |
//+------------------------------------------------------------------+
bool ChartForeColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of axes, scale and OHLC line
   if(!ChartSetInteger(chart_ID,CHART_COLOR_FOREGROUND,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

· **CHART_COLOR_GRID** chart grid color.

```
//+------------------------------------------------------------------+
//| The function receives chart grid color.                          |
//+------------------------------------------------------------------+
color ChartGridColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive chart grid color
   if(!ChartGetInteger(chart_ID,CHART_COLOR_GRID,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets chart grid color.                              |
//+------------------------------------------------------------------+
bool ChartGridColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set chart grid color
   if(!ChartSetInteger(chart_ID,CHART_COLOR_GRID,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_VOLUME** - color of volumes and order opening levels.

```
//+------------------------------------------------------------------+
//| The function receives color of volumes and market entry          |
//| levels.                                                          |
//+------------------------------------------------------------------+
color ChartVolumeColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive color of volumes and market entry levels
   if(!ChartGetInteger(chart_ID,CHART_COLOR_VOLUME,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets the color of volumes and market entry          |
//| levels.                                                          |
//+------------------------------------------------------------------+
bool ChartVolumeColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set color of volumes and market entry levels
   if(!ChartSetInteger(chart_ID,CHART_COLOR_VOLUME,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_CHART_UP** color of up bar, its shadow and border of a bullish candlestick's body.

```
//+------------------------------------------------------------------+
//| The function receives color of up bar, its shadow and            |
//| border of a bullish candlestick's body.                          |
//+------------------------------------------------------------------+
color ChartUpColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of up bar, its shadow and border of bullish candle
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_UP,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets color of up bar, its shadow and                |
//| border of a bullish candlestick's body.                          |
//+------------------------------------------------------------------+
bool ChartUpColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of up bar, its shadow and border of body of a bullish
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_UP,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_CHART_DOWN** color of down bar, its shadow and border of bearish candlestick's body.

```
//+------------------------------------------------------------------+
//| The function receives color of up bar, its shadow and            |
//| border of a bearish candlestick's body.                          |
//+------------------------------------------------------------------+
color ChartDownColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of down bar, its shadow and border of bearish canc
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_DOWN,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets color of down bar, its shadow and              |
//| border of a bearish candlestick's body.                          |
//+------------------------------------------------------------------+
bool ChartDownColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of down bar, its shadow and border of bearish candlest
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_DOWN,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

· **CHART_COLOR_CHART_LINE** color of the chart line and Doji candlesticks.

```
//+-----------------------------------------------------------------------
//| The function receives color of the chart line and Doji candlesticks.
//+-----------------------------------------------------------------------
color ChartLineColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive color of the chart line and Doji candlesticks
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_LINE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets the color of the chart line and Doji           |
//| candlesticks.                                                    |
//+------------------------------------------------------------------+
bool ChartLineColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set color of the chart line and Doji candlesticks
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_LINE,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

· **CHART_COLOR_CANDLE_BULL** color of bullish candlestick's body.

```
//+------------------------------------------------------------------+
//| The function receives color of bullish candlestick's body.      |
//+------------------------------------------------------------------+
color ChartBullColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of bullish candlestick's body
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets color of bullish candlestick's body.          |
//+------------------------------------------------------------------+
bool ChartBullColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of bullish candlestick's body
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_CANDLE_BEAR** color of bearish candlestick's body.

```
//+------------------------------------------------------------------+
//| The function receives color of bearish candlestick's body.       |
//+------------------------------------------------------------------+
color ChartBearColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of bearish candlestick's body
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets color of bearish candlestick's body.           |
//+------------------------------------------------------------------+
bool ChartBearColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of bearish candlestick's body
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_BID** Bid price line color.

```
//+------------------------------------------------------------------+
//| The function receives the color of Bid line.                     |
//+------------------------------------------------------------------+
color ChartBidColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of Bid price line
   if(!ChartGetInteger(chart_ID,CHART_COLOR_BID,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets the color of Bid line.                         |
//+------------------------------------------------------------------+
bool ChartBidColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of Bid price line
   if(!ChartSetInteger(chart_ID,CHART_COLOR_BID,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_ASK** Ask price line color.

```
//+------------------------------------------------------------------+
//| The function receives the color of Ask line.                     |
//+------------------------------------------------------------------+
color ChartAskColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of Ask price line
   if(!ChartGetInteger(chart_ID,CHART_COLOR_ASK,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets the color of Ask line.                         |
//+------------------------------------------------------------------+
bool ChartAskColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of Ask price line
   if(!ChartSetInteger(chart_ID,CHART_COLOR_ASK,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_LAST** color of the last performed deal's price line (Last).

```
//+------------------------------------------------------------------+
//| The function receives color of the last performed deal's price line. |
//+------------------------------------------------------------------+
color ChartLastColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive color of the last performed deal's price line (Last)
   if(!ChartGetInteger(chart_ID,CHART_COLOR_LAST,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets color of the last performed deal's price       |
//| line.                                                            |
//+------------------------------------------------------------------+
bool ChartLastColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set color of the last performed deal's price line (Last)
   if(!ChartSetInteger(chart_ID,CHART_COLOR_LAST,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_COLOR_STOP_LEVEL** stop order level color (Stop Loss and Take Profit).

```
//+--------------------------------------------------------------------+
//| The function receives colors of Stop Loss and Take Profit levels.  |
//+--------------------------------------------------------------------+
color ChartStopLevelColorGet(const long chart_ID=0)
  {
//--- prepare the variable to receive the color
   long result=clrNONE;
//--- reset the error value
   ResetLastError();
//--- receive the color of stop order levels (Stop Loss and Take Profit)
   if(!ChartGetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return((color)result);
  }
//+------------------------------------------------------------------+
//| The function sets Stop Loss and Take Profit level colors.        |
//+------------------------------------------------------------------+
bool ChartStopLevelColorSet(const color clr,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set the color of stop order levels (Stop Loss and Take Profit)
   if(!ChartSetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,clr))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_TRADE_LEVELS** property of displaying trade levels on the chart (levels of open orders, Stop Loss, Take Profit and pending orders).

```
//+---------------------------------------------------------------------+
//| The function defines if trading levels are displayed on the chart.  |
//+---------------------------------------------------------------------+
bool ChartShowTradeLevelsGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function enables/disables trading levels display mode.       |
//+------------------------------------------------------------------+
bool ChartShowTradeLevelsSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_DRAG_TRADE_LEVELS** property of enabling the ability to drag trading levels on a chart using mouse.

```
//+------------------------------------------------------------------
//| The function defines if dragging trading levels on a chart using mouse
//| is allowed.
//+------------------------------------------------------------------
bool ChartDragTradeLevelsGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function enables/disables the mode of dragging trade levels  |
//| on the chart using mouse.                                        |
//+------------------------------------------------------------------+
bool ChartDragTradeLevelsSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_DATE_SCALE** property of displaying the time scale on a chart.

```
//+------------------------------------------------------------------+
//| The function defines if the time scale is displayed on the chart.  |
//+------------------------------------------------------------------+
bool ChartShowDateScaleGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+---------------------------------------------------------------------
//| The function enables/disables the mode of displaying the time scale on
//| chart.
//+---------------------------------------------------------------------
bool ChartShowDateScaleSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHOW_PRICE_SCALE** property of displaying the price scale on a chart.

```
//+--------------------------------------------------------------------+
//| The function defines if the price scale is displayed on the chart. |
//+--------------------------------------------------------------------+
bool ChartShowPriceScaleGet(bool &result,const long chart_ID=0)
  {
//--- prepare the variable to get the property value
   long value;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- store the value of the chart property in memory
   result=value;
//--- successful execution
   return(true);
  }
//+-----------------------------------------------------------------------
//| The function enables/disables the mode of displaying the price scale c
//| chart.
//+-----------------------------------------------------------------------
bool ChartShowPriceScaleSet(const bool value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_SHIFT_SIZE** shift size of the zero bar from the right border in percentage values.

```
//+------------------------------------------------------------------------
//| The function receives shift size of the zero bar from the right border
//| of the chart in percentage values (from 10% up to 50%).
//+------------------------------------------------------------------------
double ChartShiftSizeGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_SHIFT_SIZE,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+------------------------------------------------------------------------
//| The function sets the shift size of the zero bar from the right
//| border of the chart in percentage values (from 10% up to 50%). To enab
//| mode, CHART_SHIFT property value should be set to
//| true.
//+------------------------------------------------------------------------
bool ChartShiftSizeSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_SHIFT_SIZE,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_IS_OFFLINE** checks offline mode of the chart.

```
//+------------------------------------------------------------------+
//| The function checks offline mode of the chart                    |
//+------------------------------------------------------------------+
bool CheckChartOffline(const long chart_ID=0)
{
   bool offline=ChartGetInteger(chart_ID,CHART_IS_OFFLINE);
   return(offline);
}
```

- **CHART_FIXED_POSITION** chart fixed position from the left border in percentage value.

```
//+-----------------------------------------------------------------------
//| The function receives the location of the chart fixed position from th
//| left border in percentage value.
//+-----------------------------------------------------------------------
double ChartFixedPositionGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_FIXED_POSITION,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+---------------------------------------------------------------------+
//| The function sets the location of the chart fixed position from the |
//| left border in percentage value. To view the location of the        |
//| chart fixed position, the value of                                  |
//| CHART_AUTOSCROLL property should be set to false.                   |
//+---------------------------------------------------------------------+
bool ChartFixedPositionSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_FIXED_POSITION,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_FIXED_MAX** property of the chart's fixed maximum.

```
//+------------------------------------------------------------------+
//| The function receives the value of chart's fixed maximum.        |
//+------------------------------------------------------------------+
double ChartFixedMaxGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_FIXED_MAX,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+------------------------------------------------------------------+
//| The function sets the value of chart's fixed maximum.            |
//| To change the value of the property,                             |
//| CHART_SCALEFIX property value should be preliminarily set to     |
//| true.                                                            |
//+------------------------------------------------------------------+
bool ChartFixedMaxSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_FIXED_MAX,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_FIXED_MIN** property of the chart's fixed minimum.

```
//+------------------------------------------------------------------+
//| The function receives the value of chart's fixed minimum.        |
//+------------------------------------------------------------------+
double ChartFixedMinGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_FIXED_MIN,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+------------------------------------------------------------------+
//| The function sets the value of chart's fixed minimum.            |
//| To change the value of the property,                             |
//| CHART_SCALEFIX property value should be preliminarily set to     |
//| true.                                                            |
//+------------------------------------------------------------------+
bool ChartFixedMinSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_FIXED_MIN,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_POINTS_PER_BAR** value of scale in points per bar.

```
//+------------------------------------------------------------------
//| The function receives the value of the chart scale in points per bar.
//+------------------------------------------------------------------
double ChartPointsPerBarGet(const long chart_ID=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_POINTS_PER_BAR,0,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
//+------------------------------------------------------------------+
//| The function sets the value of the chart scale in points per bar.    |
//| To view the result of this property's value change,                  |
//| the value of                                                         |
//| CHART_SCALE_PT_PER_BAR property should be preliminarily set to true. |
//+------------------------------------------------------------------+
bool ChartPointsPerBarSet(const double value,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetDouble(chart_ID,CHART_POINTS_PER_BAR,value))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

- **CHART_PRICE_MIN** returns the value of the chart minimum.

```
//+------------------------------------------------------------------------
//| The function receives the value of the chart minimum in the main windc
//| subwindow.
//+------------------------------------------------------------------------
double ChartPriceMin(const long chart_ID=0,const int sub_window=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_PRICE_MIN,sub_window,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
```

· **CHART_PRICE_MAX** returns the value of the chart maximum.

```
//+------------------------------------------------------------------------
//| The function receives the value of the chart maximum in the main windc
//| subwindow.
//+------------------------------------------------------------------------
double ChartPriceMax(const long chart_ID=0,const int sub_window=0)
  {
//--- prepare the variable to get the result
   double result=EMPTY_VALUE;
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetDouble(chart_ID,CHART_PRICE_MAX,sub_window,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- return the value of the chart property
   return(result);
  }
```

· **CHART_COMMENT** comment on the chart.

```
//+----------------------------------------------------------------------+
//| The function receives comment in the upper left corner of the chart. |
//+----------------------------------------------------------------------+
bool ChartCommentGet(string &result,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- receive the property value
   if(!ChartGetString(chart_ID,CHART_COMMENT,result))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function sets comment in the upper left corner of the        |
//| chart.                                                           |
//+------------------------------------------------------------------+
bool ChartCommentSet(const string str,const long chart_ID=0)
  {
//--- reset the error value
   ResetLastError();
//--- set property value
   if(!ChartSetString(chart_ID,CHART_COMMENT,str))
     {
      //--- display the error message in Experts journal
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
```

Panel for chart properties

```
//--- connect the library of control elements
#include <ChartObjects\ChartObjectsTxtControls.mqh>
//--- predefined constants
#define X_PROPERTY_NAME_1    10  // x coordinate of the property name in t
#define X_PROPERTY_VALUE_1   225 // x coordinate of the property value in
#define X_PROPERTY_NAME_2    345 // x coordinate of the property name in t
#define X_PROPERTY_VALUE_2   550 // x coordinate of the property value in
#define X_BUTTON_1           285 // x coordinate of the button in the firs
```

```
#define X_BUTTON_2           700 // x coordinate of the button in the seco
#define Y_PROPERTY_1         30  // y coordinate of the beginning of the f
#define Y_PROPERTY_2         286 // y coordinate of the beginning of the t
#define Y_DISTANCE           16  // y axial distance between the lines
#define LAST_PROPERTY_NUMBER 111 // number of the last graphical property
//--- input parameters
input color InpFirstColor=clrDodgerBlue; // Color of odd lines
input color InpSecondColor=clrGoldenrod; // Color of even lines
//--- variables and arrays
CChartObjectLabel  ExtLabelsName[];  // labels for displaying property nam
CChartObjectLabel  ExtLabelsValue[]; // labels for displaying property val
CChartObjectButton ExtButtons[];     // buttons
int                ExtNumbers[];     // property indices
string             ExtNames[];       // property names
uchar              ExtDataTypes[];   // property data types (integer, doub
uint               ExtGroupTypes[];  // array that stores the data on belo
uchar              ExtDrawTypes[];   // array that stores the data on the 
double             ExtMaxValue[];    // maximum property values that are p
double             ExtMinValue[];    // minimum property values that are p
double             ExtStep[];        // steps for changing properties
int                ExtCount;         // total number of all properties
color              ExtColors[2];     // array of colors for displaying lin
string             ExtComments[2];   // array of comments (for CHART_COMME
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- display a comment on the chart
   Comment("SomeComment");
//--- store colors in the array to be able to switch between them later
   ExtColors[0]=InpFirstColor;
   ExtColors[1]=InpSecondColor;
//--- store comments in the array to be able to switch between them later
   ExtComments[0]="FirstComment";
   ExtComments[1]="SecondComment";
//--- prepare and display the control panel for managing chart properties
   if(!PrepareControls())
      return(INIT_FAILED);
//--- successful execution
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Deinitialization function of the expert                          |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
```

```
//--- remove the comment on the chart
   Comment("");
  }
//+------------------------------------------------------------------+
//| Handler of a chart event                                         |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- check the event of clicking the chart object
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      //--- divide the object name by separator
      string obj_name[];
      StringSplit(sparam,'_',obj_name);
      //--- check if the object is a button
      if(obj_name[0]=="Button")
        {
         //--- receive button index
         int index=(int)StringToInteger(obj_name[1]);
         //--- unpress the button
         ExtButtons[index].State(false);
         //--- set the new value of the property depending on its type
         if(ExtDataTypes[index]=='I')
            ChangeIntegerProperty(index);
         if(ExtDataTypes[index]=='D')
            ChangeDoubleProperty(index);
         if(ExtDataTypes[index]=='S')
            ChangeStringProperty(index);
        }
     }
//--- re-draw property values
   RedrawProperties();
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| Change the integer property of the chart                         |
//+------------------------------------------------------------------+
void ChangeIntegerProperty(const int index)
  {
//--- receive the current property value
   long value=ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[in
//--- define the following property value
   switch(ExtDrawTypes[index])
     {
      case 'C':
         value=GetNextColor((color)value);
         break;
      default:
         value=(long)GetNextValue((double)value,index);
         break;
     }
//--- set the new property value
   ChartSetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index],0,valu
  }
//+------------------------------------------------------------------+
//| Change double property of the chart                              |
//+------------------------------------------------------------------+
void ChangeDoubleProperty(const int index)
  {
//--- receive the current property value
   double value=ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[in
//--- define the following property value
   value=GetNextValue(value,index);
//--- set the new property value
   ChartSetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index],value);
  }
//+------------------------------------------------------------------+
//| Change string property of the chart                              |
//+------------------------------------------------------------------+
void ChangeStringProperty(const int index)
  {
//--- static variable for switching inside ExtComments array
   static uint comment_index=1;
//--- change index for receiving another comment
   comment_index=1-comment_index;
//--- set the new property value
   ChartSetString(0,(ENUM_CHART_PROPERTY_STRING)ExtNumbers[index],ExtComme
  }
//+------------------------------------------------------------------+
//| Define the next property value                                   |
//+------------------------------------------------------------------+
double GetNextValue(const double value,const int index)
  {
   if(value+ExtStep[index]<=ExtMaxValue[index])
      return(value+ExtStep[index]);
   else
      return(ExtMinValue[index]);
  }
//+------------------------------------------------------------------+
//| Receive the next color for color type property                   |
//+------------------------------------------------------------------+
color GetNextColor(const color clr)
  {
//--- return the following color value
   switch(clr)
     {
      case clrWhite: return(clrRed);
      case clrRed:   return(clrGreen);
      case clrGreen: return(clrBlue);
      case clrBlue:  return(clrBlack);
      default:       return(clrWhite);
     }
  }
//+------------------------------------------------------------------+
//| Re-draw property values                                          |
//+------------------------------------------------------------------+
void RedrawProperties(void)
  {
//--- property value text
   string text;
   long   value;
//--- loop of the number of properties
   for(int i=0;i<ExtCount;i++)
     {
      text="";
      switch(ExtDataTypes[i])
        {
         case 'I':
            //--- receive the current property value
            if(!ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[
            break;
            //--- integer property text
            switch(ExtDrawTypes[i])
              {
               //--- color property
               case 'C':
                  text=(string)((color)value);
                  break;
                  //--- boolean property
               case 'B':
                  text=(string)((bool)value);
                  break;
                  //--- ENUM_CHART_MODE enumeration property
               case 'M':
                  text=EnumToString((ENUM_CHART_MODE)value);
                  break;
                  //--- ENUM_CHART_VOLUME_MODE enumeration property
               case 'V':
                  text=EnumToString((ENUM_CHART_VOLUME_MODE)value);
                  break;
                  //--- int type number
               default:
                  text=IntegerToString(value);
                  break;
              }
            break;
         case 'D':
            //--- double property text
            text=DoubleToString(ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUB
            break;
         case 'S':
            //--- string property text
            text=ChartGetString(0,(ENUM_CHART_PROPERTY_STRING)ExtNumbers[i
            break;
        }
      //--- display property value
      ExtLabelsValue[i].Description(text);
     }
  }
//+------------------------------------------------------------------+
//| Create the panel for managing chart properties                   |
//+------------------------------------------------------------------+
bool PrepareControls(void)
  {
//--- allocate memory for arrays with a reserve
   MemoryAllocation(LAST_PROPERTY_NUMBER+1);
//--- variables
   int i=0;     // loop variable
   int col_1=0; // number of properties in the first column
   int col_2=0; // number of properties in the second column
   int col_3=0; // number of properties in the third column
//--- current number of properties - 0
   ExtCount=0;
//--- looking for properties in the loop
   while(i<=LAST_PROPERTY_NUMBER)
     {
      //--- store the current number of the property
      ExtNumbers[ExtCount]=i;
      //--- increase the value of the loop variable
      i++;
      //--- check if there is a property with such a number
      if(CheckNumber(ExtNumbers[ExtCount],ExtNames[ExtCount],ExtDataTypes[
        {
         //--- create control elements for the property
         switch(ExtGroupTypes[ExtCount])
```

```
           {
            case 1:
               //--- create labels and a button for the property
               if(!ShowProperty(ExtCount,0,X_PROPERTY_NAME_1,X_PROPERTY_VA
               return(false);
               //--- number of the elements in the first column has increa
               col_1++;
               break;
            case 2:
               //--- create labels and a button for the property
               if(!ShowProperty(ExtCount,1,X_PROPERTY_NAME_2,X_PROPERTY_VA
               return(false);
               //--- number of the elements in the second column has incre
               col_2++;
               break;
            case 3:
               //--- create only labels for the property
               if(!ShowProperty(ExtCount,2,X_PROPERTY_NAME_2,X_PROPERTY_VA
               return(false);
               //--- number of the elements in the third column has increa
               col_3++;
               break;
           }
         //--- define maximum and minimum property value and step
         GetMaxMinStep(ExtNumbers[ExtCount],ExtMaxValue[ExtCount],ExtMinVa
         //--- increase the number of properties
         ExtCount++;
        }
     }
//--- free the memory not used by arrays
   MemoryAllocation(ExtCount);
//--- re-draw property values
   RedrawProperties();
   ChartRedraw();
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Allocate memory for arrays                                       |
//+------------------------------------------------------------------+
void MemoryAllocation(const int size)
  {
   ArrayResize(ExtLabelsName,size);
   ArrayResize(ExtLabelsValue,size);
   ArrayResize(ExtButtons,size);
   ArrayResize(ExtNumbers,size);
   ArrayResize(ExtNames,size);
   ArrayResize(ExtDataTypes,size);
   ArrayResize(ExtGroupTypes,size);
   ArrayResize(ExtDrawTypes,size);
   ArrayResize(ExtMaxValue,size);
   ArrayResize(ExtMinValue,size);
   ArrayResize(ExtStep,size);
  }
//+------------------------------------------------------------------+
//| Check if the property index belongs to the one of                |
//| ENUM_CHART_PROPERTIES enumerations                               |
//+------------------------------------------------------------------+
bool CheckNumber(const int ind,string &name,uchar &data_type,uint &group_t
  {
//--- check if the property is of integer type
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_INTEGER)ind);
   if(_LastError==0)
     {
      data_type='I';                      // property from ENUM_CHART_PROP
      GetTypes(ind,group_type,draw_type); // define property display param
      return(true);
     }
//--- check if the property is of double type
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_DOUBLE)ind);
   if(_LastError==0)
     {
      data_type='D';                      // property from ENUM_CHART_PROP
      GetTypes(ind,group_type,draw_type); // define property display param
      return(true);
     }
//--- check if the property is of string type
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_STRING)ind);
   if(_LastError==0)
     {
      data_type='S';                      // property from ENUM_CHART_PROP
      GetTypes(ind,group_type,draw_type); // define property display param
      return(true);
     }
//--- property does not belong to any enumeration
   return(false);
  }
//+------------------------------------------------------------------+
//| Define the group the property should be stored in,               |
//| as well as its display type                                      |
//+------------------------------------------------------------------+
void GetTypes(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- check if the property belongs to the third group
//--- third group properties are displayed in the second column starting f
   if(CheckThirdGroup(property_number,group_type,draw_type))
      return;
//--- check if the property belongs to the second group
//--- second group properties are displayed at the beginning of the second
   if(CheckSecondGroup(property_number,group_type,draw_type))
      return;
//--- if you find yourself here, the property belongs to the first group (
   CheckFirstGroup(property_number,group_type,draw_type);
  }
//+-----------------------------------------------------------------------+
//| The function checks if the property belongs to the third group and   |
//| defines its display type in case of a positive answer                 |
//+-----------------------------------------------------------------------+
bool CheckThirdGroup(const int property_number,uint &group_type,uchar &dra
  {
//--- check if the property belongs to the third group
   switch(property_number)
     {
      //--- boolean properties
      case CHART_WINDOW_IS_VISIBLE:
         draw_type='B';
         break;
         //--- integer properties
      case CHART_VISIBLE_BARS:
      case CHART_WINDOWS_TOTAL:
      case CHART_WINDOW_HANDLE:
      case CHART_WINDOW_YDISTANCE:
      case CHART_FIRST_VISIBLE_BAR:
      case CHART_WIDTH_IN_BARS:
      case CHART_WIDTH_IN_PIXELS:
         draw_type='I';
         break;
         //--- double properties
      case CHART_PRICE_MIN:
      case CHART_PRICE_MAX:
         draw_type='D';
         break;
         //--- in fact, this property is a command of displaying the chart
         //--- there is no need to apply this panel, as the window will al
         //--- on top of other ones before we use it
      case CHART_BRING_TO_TOP:
         draw_type=' ';
         break;
         //--- property does not belong to the third group
      default:
         return(false);
     }
//--- property belongs to the third group
   group_type=3;
   return(true);
  }
//+---------------------------------------------------------------------+
//| The function checks if the property belongs to the second group and |
//| defines its display type in case of a positive answer           |
//+---------------------------------------------------------------------+
bool CheckSecondGroup(const int property_number,uint &group_type,uchar &dr
  {
//--- check if the property belongs to the second group
   switch(property_number)
     {
      //--- ENUM_CHART_MODE type property
      case CHART_MODE:
         draw_type='M';
         break;
         //--- ENUM_CHART_VOLUME_MODE type property
      case CHART_SHOW_VOLUMES:
         draw_type='V';
         break;
         //--- string property
      case CHART_COMMENT:
         draw_type='S';
         break;
         //--- color property
      case CHART_COLOR_BACKGROUND:
      case CHART_COLOR_FOREGROUND:
      case CHART_COLOR_GRID:
      case CHART_COLOR_VOLUME:
      case CHART_COLOR_CHART_UP:
      case CHART_COLOR_CHART_DOWN:
      case CHART_COLOR_CHART_LINE:
      case CHART_COLOR_CANDLE_BULL:
      case CHART_COLOR_CANDLE_BEAR:
      case CHART_COLOR_BID:
      case CHART_COLOR_ASK:
      case CHART_COLOR_LAST:
      case CHART_COLOR_STOP_LEVEL:
         draw_type='C';
         break;
         //--- property does not belong to the second group
      default:
         return(false);
     }
//--- property belongs to the second group
   group_type=2;
   return(true);
  }
//+-----------------------------------------------------------------------
//| This function is called only if it is already known that
//| the property does not belong to the second and third property groups
//+-----------------------------------------------------------------------
void CheckFirstGroup(const int property_number,uint &group_type,uchar &dra
  {
//--- the property belongs to the first group
   group_type=1;
//--- define property display type
   switch(property_number)
     {
      //--- integer properties
      case CHART_SCALE:
      case CHART_HEIGHT_IN_PIXELS:
         draw_type='I';
         return;
         //--- double properties
      case CHART_SHIFT_SIZE:
      case CHART_FIXED_POSITION:
      case CHART_FIXED_MAX:
      case CHART_FIXED_MIN:
      case CHART_POINTS_PER_BAR:
         draw_type='D';
         return;
         //--- only boolean properties have remained
      default:
         draw_type='B';
         return;
     }
  }
//+------------------------------------------------------------------+
//| Create a label and a button for the property                     |
//+------------------------------------------------------------------+
bool ShowProperty(const int ind,const int type,const int x1,const int x2,
                  const int xb,const int y,const bool btn)
  {
//--- static array for switching inside ExtColors color array
   static uint color_index[3]={1,1,1};
//--- change index for receiving another color
   color_index[type]=1-color_index[type];
//--- display labels and a button (if btn=true) for the property
```

```
   if(!LabelCreate(ExtLabelsName[ind],"name_"+(string)ind,ExtNames[ind],Ex
      return(false);
   if(!LabelCreate(ExtLabelsValue[ind],"value_"+(string)ind,"",ExtColors[c
      return(false);
   if(btn && !ButtonCreate(ExtButtons[ind],(string)ind,xb,y+1))
      return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Create a label                                                   |
//+------------------------------------------------------------------+
bool LabelCreate(CChartObjectLabel &lbl,const string name,const string tex
                 const color clr,const int x,const int y)
  {
   if(!lbl.Create(0,"Label_"+name,0,x,y)) return(false);
   if(!lbl.Description(text))             return(false);
   if(!lbl.FontSize(10))                  return(false);
   if(!lbl.Color(clr))                    return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the button                                                |
//+------------------------------------------------------------------+
bool ButtonCreate(CChartObjectButton &btn,const string name,
                  const int x,const int y)
  {
   if(!btn.Create(0,"Button_"+name,0,x,y,50,15)) return(false);
   if(!btn.Description("Next"))                   return(false);
   if(!btn.FontSize(10))                          return(false);
   if(!btn.Color(clrBlack))                       return(false);
   if(!btn.BackColor(clrWhite))                   return(false);
   if(!btn.BorderColor(clrBlack))                 return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Define maximum and minimum property value and step               |
//+------------------------------------------------------------------+
void GetMaxMinStep(const int property_number,double &max,double &min,doubl
  {
   double value;
//--- set values depending on the property type
   switch(property_number)
     {
      case CHART_SCALE:
         max=5;
         min=0;
         step=1;
         break;
      case CHART_MODE:
      case CHART_SHOW_VOLUMES:
         max=2;
         min=0;
         step=1;
         break;
      case CHART_SHIFT_SIZE:
         max=50;
         min=10;
         step=2.5;
         break;
      case CHART_FIXED_POSITION:
         max=90;
         min=0;
         step=15;
         break;
      case CHART_POINTS_PER_BAR:
         max=19;
         min=1;
         step=3;
         break;
      case CHART_FIXED_MAX:
         value=ChartGetDouble(0,CHART_FIXED_MAX);
         max=value*1.25;
         min=value;
         step=value/32;
         break;
      case CHART_FIXED_MIN:
         value=ChartGetDouble(0,CHART_FIXED_MIN);
         max=value;
         min=value*0.75;
         step=value/32;
         break;
      case CHART_HEIGHT_IN_PIXELS:
         max=700;
         min=520;
         step=30;
         break;
         //--- default values
      default:
         max=1;
         min=0;
         step=1;
```

```
        }
    }
```

Object Constants

There are 40 graphical objects that can be created and displayed in the price chart. All constants for working with objects are divided into 9 groups:

- Object types Identifiers of graphical objects;
- Object properties setting and getting properties of graphical objects;
- Methods of object binding constants of object positioning in the chart;
- Binding corner setting the corner relative to which an object is positioned on chart;
- Visibility of objects setting timeframes in which an object is visible;
- Gann objects trend constants for Gann fan and Gann grid;
- Web colors constants of predefined web colors;
- Wingdings codes of characters of the Wingdings font.

Object Types

When a graphical object is created using the ObjectCreate() function, it's necessary to specify the type of object being created, which can be one of the values of the ENUM_OBJECT enumeration. Object type identifiers are used in ObjectCreate(), ObjectsDeleteAll() and ObjectType() functions.

Further specifications of object properties are possible using functions for working with graphical objects.

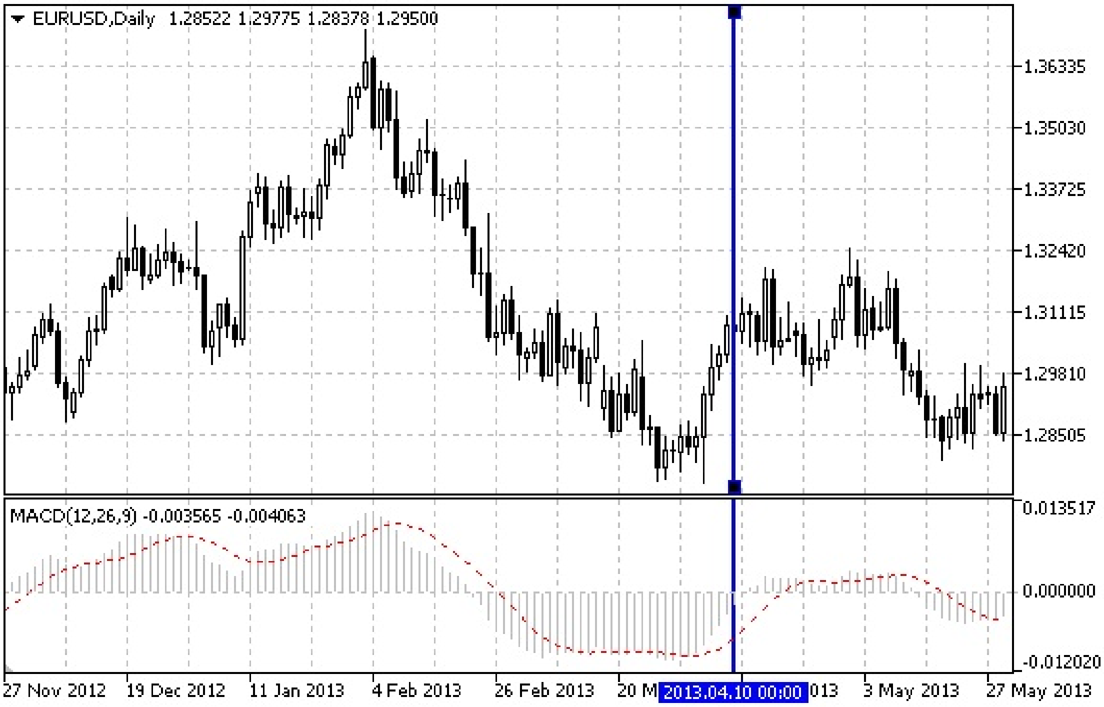
ENUM_OBJECT

ID		Description
OBJ_VLINE	|	Vertical Line
OBJ_HLINE	—	Horizontal Line
OBJ_TREND	/	Trend Line
OBJ_TRENDBYANGLE	∠	Trend Line By Angle
OBJ_CYCLES	⁞	Cycle Lines
OBJ_CHANNEL	E	Equidistant Channel
OBJ_STDDEVCHANNEL	D	Standard Deviation Channel
OBJ_REGRESSION	⋰	Linear Regression Channel
OBJ_PITCHFORK	⫽	Andrews Pitchfork
OBJ_GANNLINE	G	Gann Line
OBJ_GANNFAN	G	Gann Fan
OBJ_GANNGRID	G	Gann Grid
OBJ_FIBO	F	Fibonacci Retracement
OBJ_FIBOTIMES	F	Fibonacci Time Zones
OBJ_FIBOFAN	F	Fibonacci Fan
OBJ_FIBOARC	F	Fibonacci Arcs
OBJ_FIBOCHANNEL	F	Fibonacci Channel
OBJ_EXPANSION	F	Fibonacci Expansion
OBJ_RECTANGLE	■	Rectangle
OBJ_TRIANGLE	▲	Triangle
OBJ_ELLIPSE	●	Ellipse

OBJ_ARROW_THUMB_UP	👍	Thumbs Up
OBJ_ARROW_THUMB_DOWN	👎	Thumbs Down
OBJ_ARROW_UP	⇧	Arrow Up
OBJ_ARROW_DOWN	⇩	Arrow Down
OBJ_ARROW_STOP	⛔	Stop Sign
OBJ_ARROW_CHECK	✅	Check Sign
OBJ_ARROW_LEFT_PRICE	▭⊣	Left Price Label
OBJ_ARROW_RIGHT_PRICE	⊢▭	Right Price Label
OBJ_ARROW_BUY	⇧	Buy Sign
OBJ_ARROW_SELL	⇩	Sell Sign
OBJ_ARROW	✧	Arrow
OBJ_TEXT	T	Text
OBJ_LABEL	A	Label
OBJ_BUTTON	OK	Button
OBJ_BITMAP	🖼	Bitmap
OBJ_BITMAP_LABEL	🖼	Bitmap Label
OBJ_EDIT	I	Edit
OBJ_EVENT	13:30	The "Event" object corresponding to an event in the economic calendar
OBJ_RECTANGLE_LABEL	▬	The "Rectangle label" object for creating and designing the custom graphical interface.

OBJ_VLINE

Vertical Line.



Example

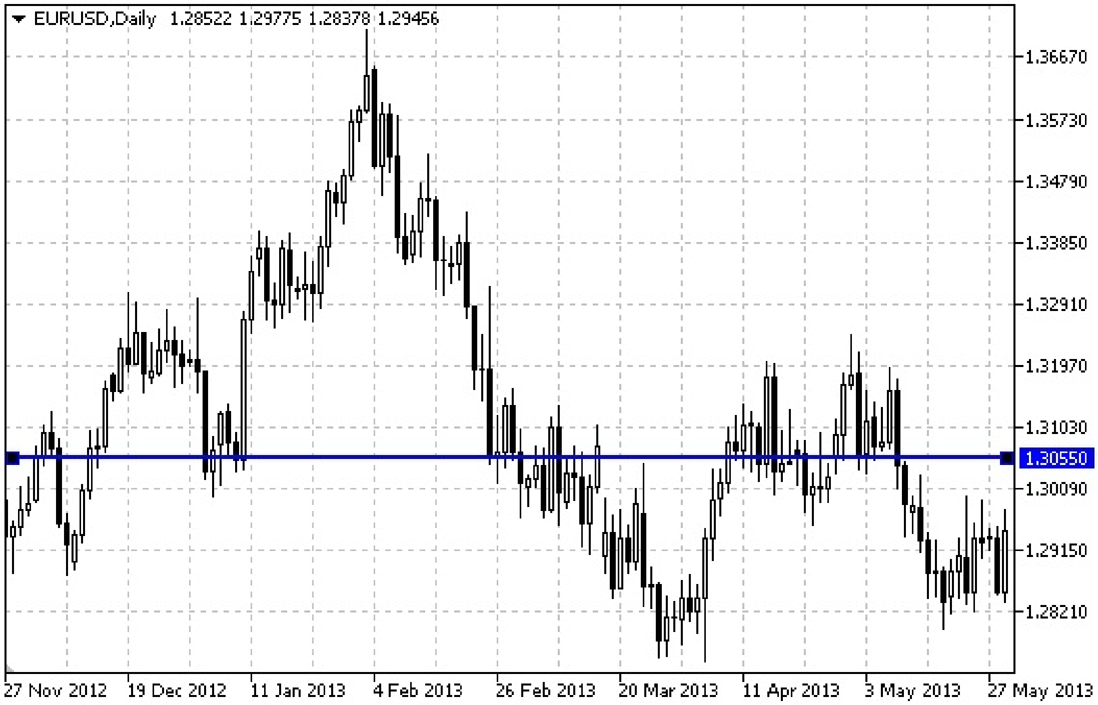
The following script creates and moves the vertical line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Vertical Line\" graphical object."
#property description "Anchor point date is set in percentage of"
#property description "the chart window width in bars."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="VLine";      // Line name
input int             InpDate=25;           // Event date, %
input color           InpColor=clrRed;      // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int             InpWidth=1;           // Line width
input bool            InpBack=false;        // Background line
input bool            InpSelection=true;    // Highlight to move
input bool            InpHidden=true;       // Hidden in the object list
input long            InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the vertical line                                         |
//+------------------------------------------------------------------+
bool VLineCreate(const long            chart_ID=0,        // chart's ID
                 const string          name="VLine",      // line name
                 const int             sub_window=0,      // subwindow ind
                 datetime              time=0,            // line time
                 const color           clr=clrRed,        // line color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                 const int             width=1,           // line width
                 const bool            back=false,        // in the backgr
                 const bool            selection=true,    // highlight to
                 const bool            hidden=true,       // hidden in the
                 const long            z_order=0)         // priority for
  {
//--- if the line time is not set, draw it via the last bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- create a vertical line
   if(!ObjectCreate(chart_ID,name,OBJ_VLINE,sub_window,time,0))
     {
      Print(__FUNCTION__,
            ": failed to create a vertical line! Error code = ",GetLastErr
      return(false);
     }
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mous
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the vertical line                                           |
//+------------------------------------------------------------------+
bool VLineMove(const long   chart_ID=0,   // chart's ID
               const string name="VLine", // line name
               datetime     time=0)       // line time
  {
//--- if line time is not set, move the line to the last bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- move the vertical line
   if(!ObjectMove(chart_ID,name,0,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move the vertical line! Error code = ",GetLastErr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the vertical line                                         |
//+------------------------------------------------------------------+
bool VLineDelete(const long   chart_ID=0,   // chart's ID
                 const string name="VLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete the vertical line
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the vertical line! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- array for storing the date values to be used
//--- for setting and changing line anchor point's coordinates
   datetime date[];
//--- memory allocation
   ArrayResize(date,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- define points for drawing the line
   int d=InpDate*(bars-1)/100;
//--- create a vertical line
   if(!VLineCreate(0,InpName,0,date[d],InpColor,InpStyle,InpWidth,InpBack,
      InpSelection,InpHidden,InpZOrder))
      return;
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the line
//--- loop counter
   int h_steps=bars/2;
//--- move the line
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!VLineMove(0,InpName,date[d]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.03 seconds of delay
      Sleep(30);
     }
//--- 1 second of delay
   Sleep(1000);
```

```
//--- delete the channel from the chart
   VLineDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_HLINE

Horizontal Line.



Example

The following script creates and moves the horizontal line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Horizontal Line\" graphical object."
#property description "Anchor point price is set in percentage of the heig
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="HLine";      // Line name
input int             InpPrice=25;          // Line price, %
input color           InpColor=clrRed;      // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int             InpWidth=1;           // Line width
input bool            InpBack=false;        // Background line
input bool            InpSelection=true;    // Highlight to move
input bool            InpHidden=true;       // Hidden in the object list
input long            InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the horizontal line                                       |
//+------------------------------------------------------------------+
bool HLineCreate(const long            chart_ID=0,        // chart's ID
                 const string          name="HLine",      // line name
                 const int             sub_window=0,      // subwindow ind
                 double                price=0,           // line price
                 const color           clr=clrRed,        // line color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                 const int             width=1,           // line width
                 const bool            back=false,        // in the backgr
                 const bool            selection=true,    // highlight to
                 const bool            hidden=true,       // hidden in the
                 const long            z_order=0)         // priority for
  {
//--- if the price is not set, set it at the current Bid price level
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- create a horizontal line
   if(!ObjectCreate(chart_ID,name,OBJ_HLINE,sub_window,0,price))
     {
      Print(__FUNCTION__,
            ": failed to create a horizontal line! Error code = ",GetLastE
      return(false);
     }
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mous
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
```

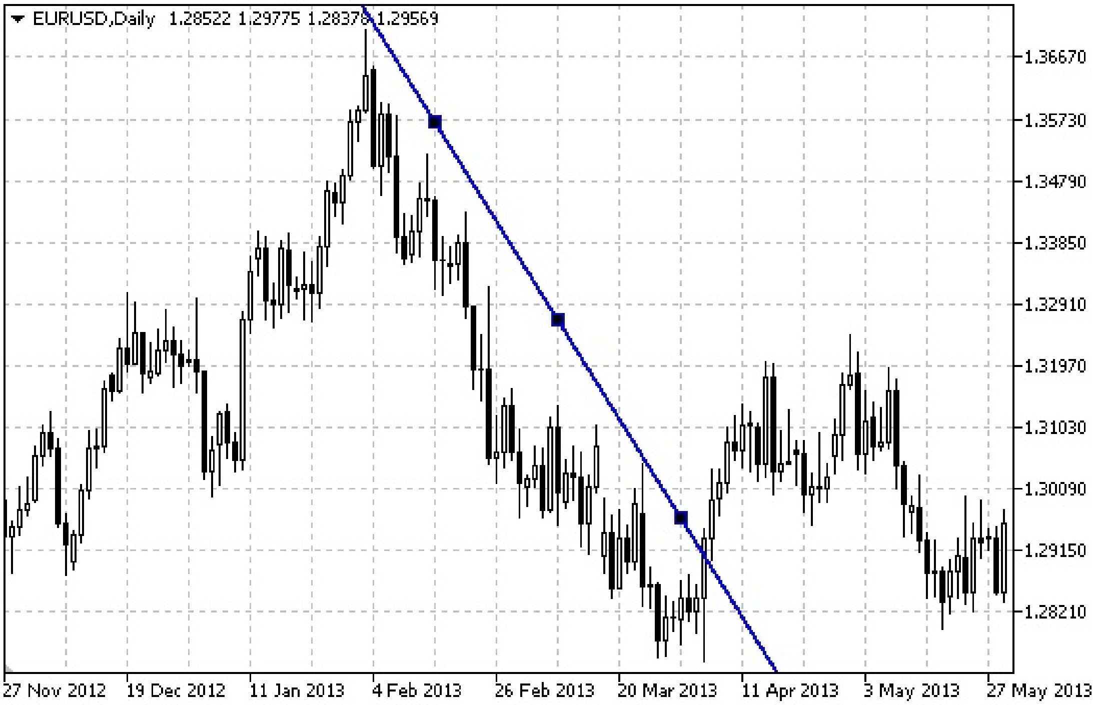
```
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move horizontal line                                             |
//+------------------------------------------------------------------+
bool HLineMove(const long   chart_ID=0,   // chart's ID
               const string name="HLine", // line name
               double       price=0)      // line price
  {
//--- if the line price is not set, move it to the current Bid price level
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move a horizontal line
   if(!ObjectMove(chart_ID,name,0,0,price))
     {
      Print(__FUNCTION__,
            ": failed to move the horizontal line! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete a horizontal line                                         |
//+------------------------------------------------------------------+
bool HLineDelete(const long   chart_ID=0,   // chart's ID
                 const string name="HLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete a horizontal line
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a horizontal line! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
//--- check correctness of the input parameters
   if(InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- price array size
   int accuracy=1000;
//--- array for storing the price values to be used
//--- for setting and changing line anchor point's coordinates
   double price[];
//--- memory allocation
   ArrayResize(price,accuracy);
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the line
   int p=InpPrice*(accuracy-1)/100;
//--- create a horizontal line
   if(!HLineCreate(0,InpName,0,price[p],InpColor,InpStyle,InpWidth,InpBack
      InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the line
//--- loop counter
   int v_steps=accuracy/2;
//--- move the line
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p<accuracy-1)
         p+=1;
      //--- move the point
      if(!HLineMove(0,InpName,price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
```

```
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete from the chart
   HLineDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_TREND

Trend Line.



Note

For Trend Line, it is possible to specify the mode of continuation of its display to the right (OBJPROP_RAY_RIGHT property).

Example

The following script creates and moves the trend line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Trend Line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Trend";     // Line name
input int             InpDate1=35;         // 1 st point's date, %
input int             InpPrice1=60;        // 1 st point's price, %
input int             InpDate2=65;         // 2 nd point's date, %
input int                InpPrice2=40;        // 2 nd point's price, %
input color              InpColor=clrRed;     // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int                InpWidth=1;          // Line width
input bool               InpBack=false;       // Background line
input bool               InpSelection=true;   // Highlight to move
input bool               InpRayRight=false;   // Line's continuation to the r
input bool               InpHidden=true;      // Hidden in the object list
input long               InpZOrder=0;         // Priority for mouse click
//+------------------------------------------------------------------+
//| Create a trend line by the given coordinates                     |
//+------------------------------------------------------------------+
bool TrendCreate(const long            chart_ID=0,        // chart's ID
                 const string          name="TrendLine",  // line name
                 const int             sub_window=0,      // subwindow ind
                 datetime              time1=0,           // first point t
                 double                price1=0,          // first point p
                 datetime              time2=0,           // second point
                 double                price2=0,          // second point
                 const color           clr=clrRed,        // line color
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                 const int             width=1,           // line width
                 const bool            back=false,        // in the backgr
                 const bool            selection=true,    // highlight to
                 const bool            ray_right=false,   // line's contin
                 const bool            hidden=true,       // hidden in the
                 const long            z_order=0)         // priority for
  {
//--- set anchor points' coordinates if they are not set
   ChangeTrendEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create a trend line by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_TREND,sub_window,time1,price1,time2,
     {
      Print(__FUNCTION__,
            ": failed to create a trend line! Error code = ",GetLastError(
      return(false);
     }
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

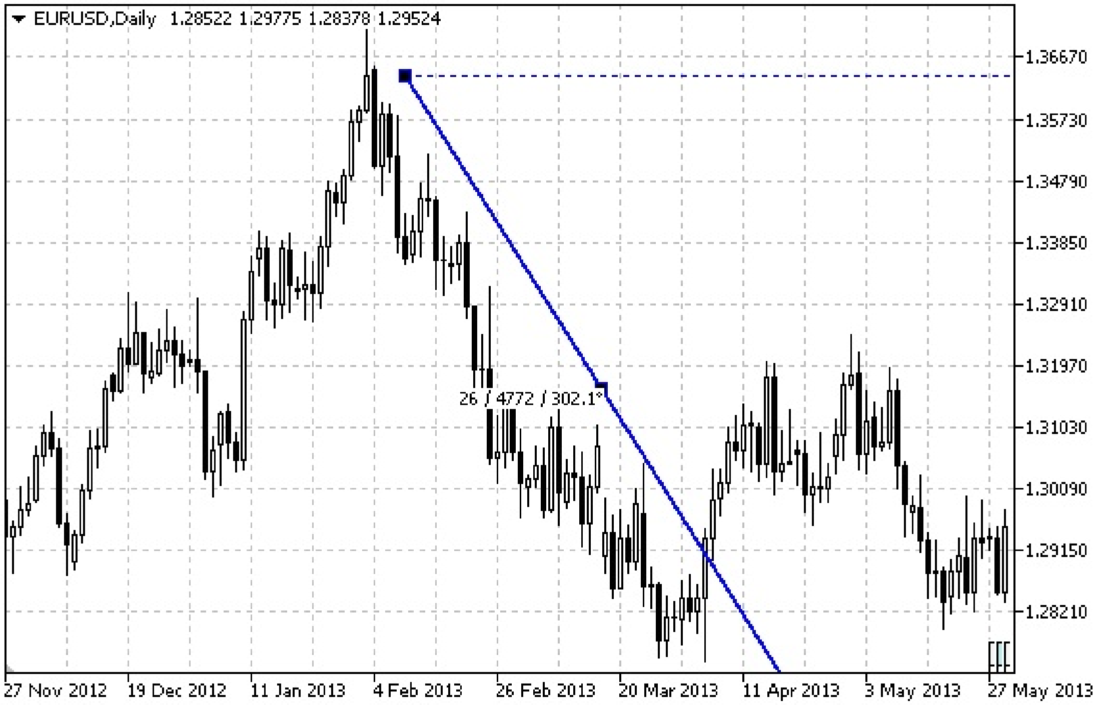
```
//--- enable (true) or disable (false) the mode of moving the line by mous
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the lin
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move trend line anchor point                                     |
//+------------------------------------------------------------------+
bool TrendPointChange(const long   chart_ID=0,       // chart's ID
                      const string name="TrendLine", // line name
                      const int    point_index=0,    // anchor point index
                      datetime     time=0,           // anchor point time
                      double       price=0)          // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move trend line's anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function deletes the trend line from the chart.              |
//+------------------------------------------------------------------+
bool TrendDelete(const long   chart_ID=0,       // chart's ID
                 const string name="TrendLine") // line name
  {
```

```
//--- reset the error value
   ResetLastError();
//--- delete a trend line
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a trend line! Error code = ",GetLastError(
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of trend line's anchor points and set default   |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left fro
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, it is equal to the first poi
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing line anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the line
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create a trend line
   if(!TrendCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpCo
      InpWidth,InpBack,InpSelection,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the line's anchor points
//--- loop counter
   int v_steps=accuracy/5;
//--- move the first anchor point vertically
```

```
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- move the second anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2<accuracy-1)
         p2+=1;
      //--- move the point
      if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- half a second of delay
   Sleep(500);
//--- loop counter
   int h_steps=bars/2;
//--- move both anchor points horizontally at the same time
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following values
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- shift the points
      if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.03 seconds of delay
      Sleep(30);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete a trend line
   TrendDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_TRENDBYANGLE

Trend Line By Angle.



Note

For Trend Line By Angle, it is possible to specify the mode of continuation of its display to the right (OBJPROP_RAY_RIGHT property).

Both angle and the second anchor point's coordinates can be used to set the slope of the line.

Example

The following script creates and moves the trend line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Trend Line By Angle\" graphical obje
#property description "Anchor point coordinates are set in percentage of t
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Trend";     // Line name
input int             InpDate1=50;         // 1 st point's date, %
input int                   InpPrice1=75;         // 1 st point's price, %
input int                   InpAngle=0;           // Line's slope angle
input color                 InpColor=clrRed;      // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Line style
input int                   InpWidth=1;           // Line width
input bool                  InpBack=false;        // Background line
input bool                  InpSelection=true;    // Highlight to move
input bool                  InpRayRight=true;     // Line's continuation to the r
input bool                  InpHidden=true;       // Hidden in the object list
input long                  InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create a trend line by angle                                     |
//+------------------------------------------------------------------+
bool TrendByAngleCreate(const long            chart_ID=0,        // chart'
                        const string          name="TrendLine",  // line n
                        const int             sub_window=0,      // subwin
                        datetime              time=0,            // point
                        double                price=0,           // point
                        const double          angle=45.0,        // slope
                        const color           clr=clrRed,        // line c
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // line s
                        const int             width=1,           // line w
                        const bool            back=false,        // in the
                        const bool            selection=true,    // highli
                        const bool            ray_right=true,    // line's
                        const bool            hidden=true,       // hidden
                        const long            z_order=0)         // priori
  {
//--- create the second point to facilitate dragging the trend line by mou
   datetime time2=0;
   double   price2=0;
//--- set anchor points' coordinates if they are not set
   ChangeTrendEmptyPoints(time,price,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create a trend line using 2 points
   if(!ObjectCreate(chart_ID,name,OBJ_TRENDBYANGLE,sub_window,time,price,t
     {
      Print(__FUNCTION__,
            ": failed to create a trend line! Error code = ",GetLastError(
      return(false);
     }
//--- change trend line's slope angle; when changing the angle, coordinate
//--- point of the line are redefined automatically according to the angle
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
```

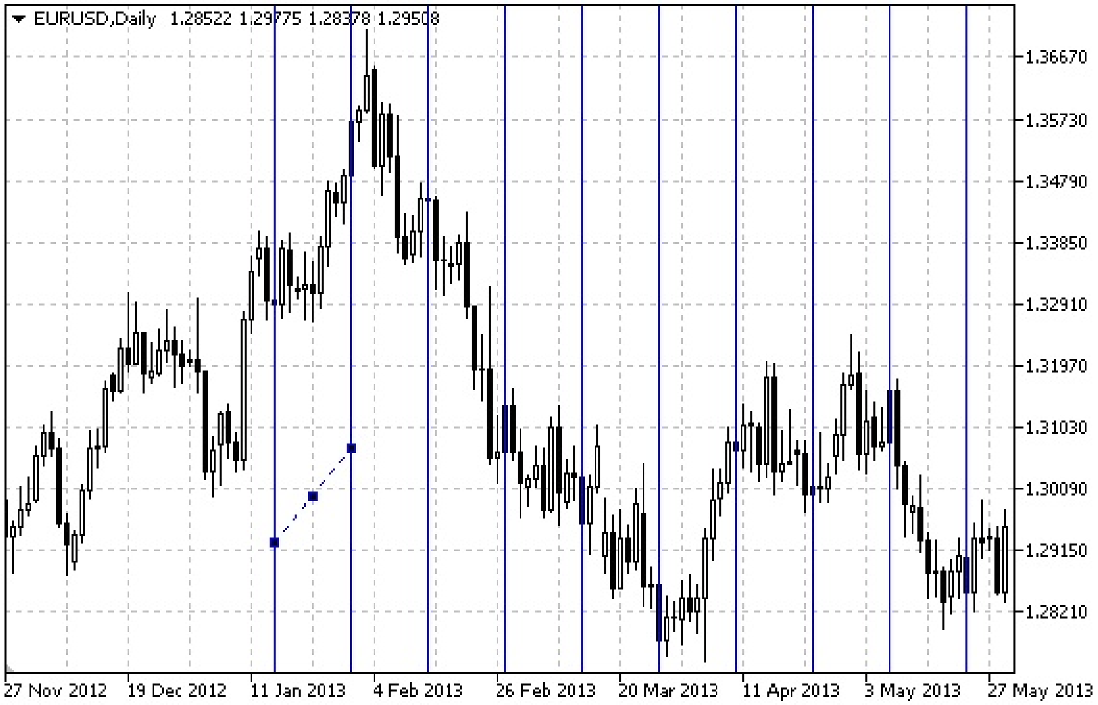
```
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the line by mous
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the lin
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change trend line anchor point's coordinates                     |
//+------------------------------------------------------------------+
bool TrendPointChange(const long   chart_ID=0,       // chart's ID
                      const string name="TrendLine", // line name
                      datetime     time=0,           // anchor point time
                      double       price=0)          // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move trend line's anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change trend line's slope angle                                   |
//+-------------------------------------------------------------------+
bool TrendAngleChange(const long   chart_ID=0,       // chart's ID
                      const string name="TrendLine", // trend line name
                      const double angle=45)         // trend line's slope
  {
//--- reset the error value
   ResetLastError();
//--- change trend line's slope angle
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
     {
      Print(__FUNCTION__,
            ": failed to change the line's slope angle! Error code = ",Get
      return(false);
     }
//--- successful execution
   return(true);
  }
//+-------------------------------------------------------------------+
//| Delete the trend line                                             |
//+-------------------------------------------------------------------+
bool TrendDelete(const long   chart_ID=0,       // chart's ID
                 const string name="TrendLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete a trend line
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a trend line! Error code = ",GetLastError(
      return(false);
     }
//--- successful execution
   return(true);
  }
//+-------------------------------------------------------------------+
//| Check the values of trend line's anchor points and set default    |
//| values for empty ones                                             |
//+-------------------------------------------------------------------+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
```

```
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- set coordinates of the second, auxiliary point
//--- the second point will be 9 bars left and have the same price
   datetime second_point_time[10];
   CopyTime(Symbol(),Period(),time1,10,second_point_time);
   time2=second_point_time[0];
   price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing line anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the line
```

```
   int d1=InpDate1*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- create a trend line
   if(!TrendByAngleCreate(0,InpName,0,date[d1],price[p1],InpAngle,InpColor
      InpWidth,InpBack,InpSelection,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move and rotate the line
//--- loop counter
   int v_steps=accuracy/2;
//--- move the anchor point and change the line's slope angle
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!TrendPointChange(0,InpName,date[d1],price[p1]))
         return;
      if(!TrendAngleChange(0,InpName,18*(i+1)))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete from the chart
   TrendDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_CYCLES

Cycle Lines.



Note

The distance between the lines is set by time coordinates of two anchor points of the object.

Example

The following script creates and moves cycle lines on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates cycle lines on the chart."
#property description "Anchor point coordinates are set in percentage"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Cycles";   // Object name
input int             InpDate1=10;        // 1 st point's date, %
input int             InpPrice1=45;       // 1 st point's price, %
input int             InpDate2=20;        // 2 nd point's date, %
input int             InpPrice2=55;       // 2 nd point's price, %
input color           InpColor=clrRed;    // Color of cycle lines
input ENUM_LINE_STYLE InpStyle=STYLE_DOT; // Style of cycle lines
input int             InpWidth=1;         // Width of cycle lines
input bool            InpBack=false;      // Background object
input bool            InpSelection=true;  // Highlight to move
input bool            InpHidden=true;     // Hidden in the object list
input long            InpZOrder=0;        // Priority for mouse click
//+------------------------------------------------------------------+
//| Create cycle lines                                               |
//+------------------------------------------------------------------+
bool CyclesCreate(const long            chart_ID=0,        // chart's ID
                  const string          name="Cycles",     // object name
                  const int             sub_window=0,      // subwindow in
                  datetime              time1=0,           // first point
                  double                price1=0,          // first point
                  datetime              time2=0,           // second point
                  double                price2=0,          // second point
                  const color           clr=clrRed,        // color of cyc
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // style of cyc
                  const int             width=1,           // width of cyc
                  const bool            back=false,        // in the backg
                  const bool            selection=true,    // highlight to
                  const bool            hidden=true,       // hidden in th
                  const long            z_order=0)         // priority for
  {
//--- set anchor points' coordinates if they are not set
   ChangeCyclesEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create cycle lines by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_CYCLES,sub_window,time1,price1,time2
     {
      Print(__FUNCTION__,
            ": failed to create cycle lines! Error code = ",GetLastError()
      return(false);
     }
//--- set color of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set display style of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the lines by mou
//--- when creating a graphical object using ObjectCreate function, the ob
```

```
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool CyclesPointChange(const long   chart_ID=0,    // chart's ID
                       const string name="Cycles", // object name
                       const int    point_index=0, // anchor point index
                       datetime     time=0,        // anchor point time co
                       double       price=0)       // anchor point price c
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the cycle lines                                           |
//+------------------------------------------------------------------+
bool CyclesDelete(const long   chart_ID=0,    // chart's ID
                  const string name="Cycles") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete cycle lines
   if(!ObjectDelete(chart_ID,name))
```

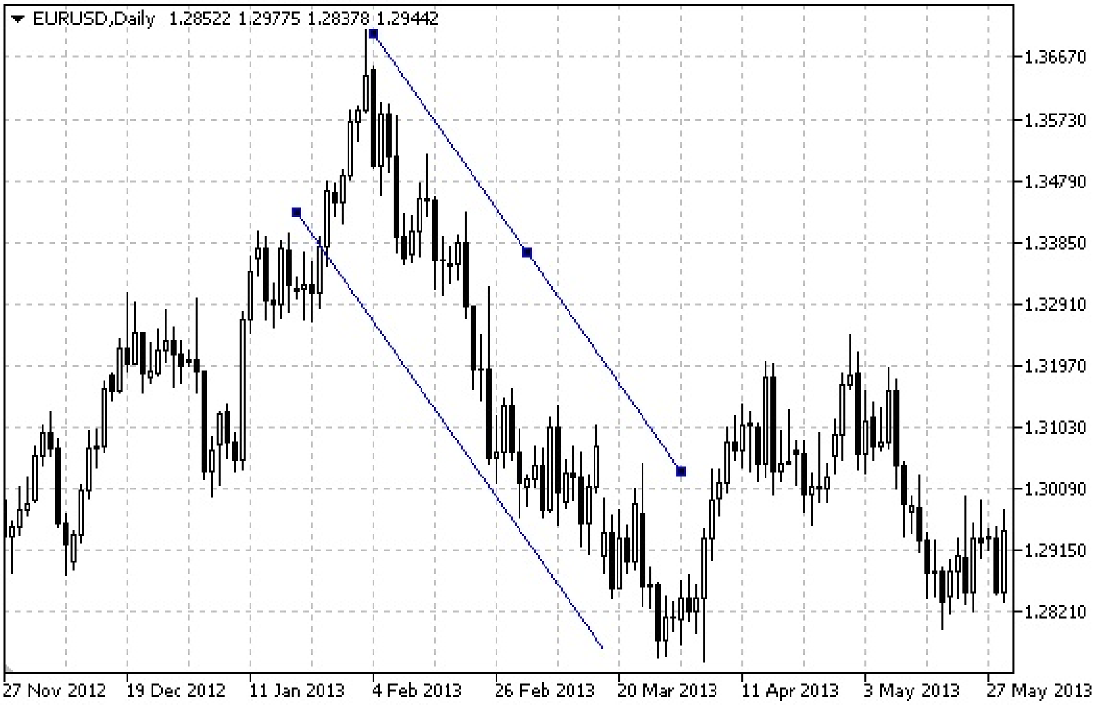
```
     {
      Print(__FUNCTION__,
            ": failed to delete cycle lines! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------
//| Check the values of cycle lines' anchor points and set default values
//| values for empty ones
//+------------------------------------------------------------------
void ChangeCyclesEmptyPoints(datetime &time1,double &price1,
                             datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left fro
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, it is equal to the first poi
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
```

```
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of cycle lines' anchor poin
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing cycle lines
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create a trend line
   if(!CyclesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpC
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int h_steps=bars/5;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
```

```
         d2+=1;
      //--- move the point
      if(!CyclesPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   h_steps=bars/4;
//--- move the first anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d1<bars-1)
         d1+=1;
      //--- move the point
      if(!CyclesPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   CyclesDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_CHANNEL

Equidistant Channel



Note

For an equidistant channel, it is possible to specify the mode of its continuation to the right (OBJPROP_RAY_RIGHT property). The mode of filling the channel with color can also be set.

Example

The following script creates and moves an equidistant channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Equidistant Channel\" graphical obje
#property description "Anchor point coordinates are set in percentage of t
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Channel";     // Channel name
input int             InpDate1=25;           // 1 st point's date, %
input int             InpPrice1=60;        // 1 st point's price, %
input int             InpDate2=65;         // 2 nd point's date, %
input int             InpPrice2=80;        // 2 nd point's price, %
input int             InpDate3=30;         // 3 rd point's date, %
input int             InpPrice3=40;        // 3 rd point's date, %
input color           InpColor=clrRed;     // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Style of channel lines
input int             InpWidth=1;          // Channel line width
input bool            InpBack=false;       // Background channel
input bool            InpFill=false;       // Filling the channel with col
input bool            InpSelection=true;   // Highlight to move
input bool            InpRayRight=false;   // Channel's continuation to th
input bool            InpHidden=true;      // Hidden in the object list
input long            InpZOrder=0;         // Priority for mouse click
//+------------------------------------------------------------------+
//| Create an equidistant channel by the given coordinates          |
//+------------------------------------------------------------------+
bool ChannelCreate(const long            chart_ID=0,        // chart's ID
                   const string          name="Channel",    // channel nam
                   const int             sub_window=0,      // subwindow i
                   datetime              time1=0,           // first point
                   double                price1=0,          // first point
                   datetime              time2=0,           // second poin
                   double                price2=0,          // second poin
                   datetime              time3=0,           // third point
                   double                price3=0,          // third point
                   const color           clr=clrRed,        // channel col
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // style of ch
                   const int             width=1,           // width of ch
                   const bool            fill=false,        // filling the
                   const bool            back=false,        // in the back
                   const bool            selection=true,    // highlight t
                   const bool            ray_right=false,   // channel's c
                   const bool            hidden=true,       // hidden in t
                   const long            z_order=0)         // priority fo
  {
//--- set anchor points' coordinates if they are not set
   ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- create a channel by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_CHANNEL,sub_window,time1,price1,time
     {
      Print(__FUNCTION__,
            ": failed to create an equidistant channel! Error code = ",Get
      return(false);
     }
```

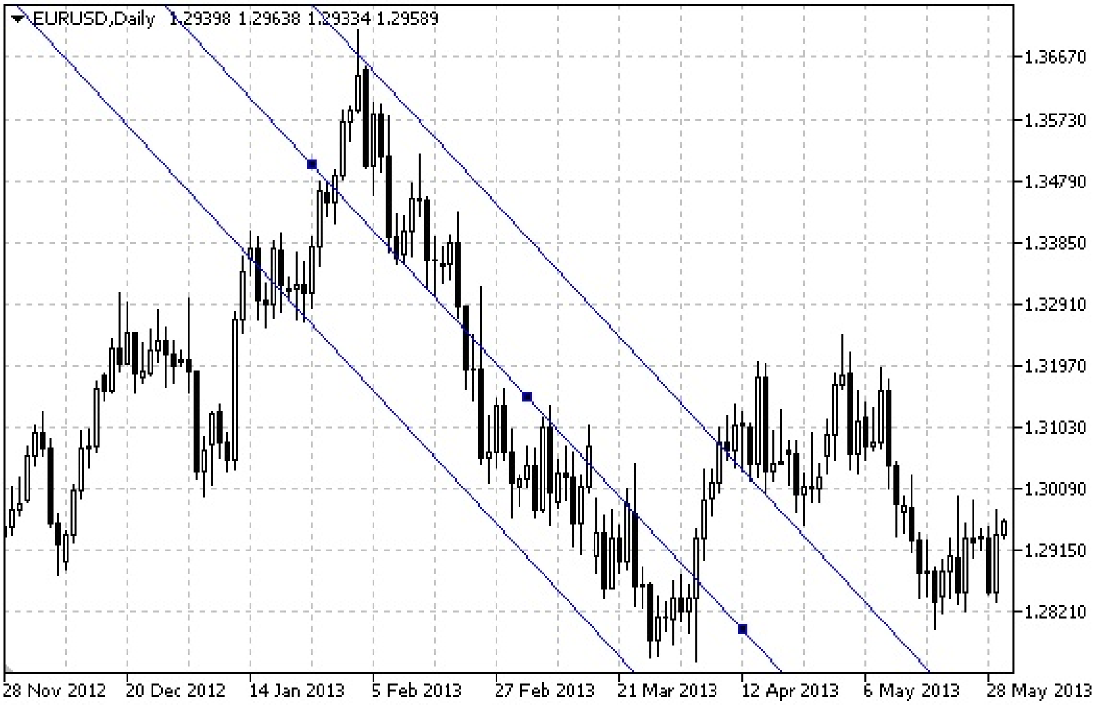
```
//--- set channel color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channe
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the channel's anchor point                                  |
//+------------------------------------------------------------------+
bool ChannelPointChange(const long   chart_ID=0,     // chart's ID
                        const string name="Channel", // channel name
                        const int    point_index=0,  // anchor point index
                        datetime     time=0,         // anchor point time
                        double       price=0)        // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the channel                                               |
//+------------------------------------------------------------------+
bool ChannelDelete(const long   chart_ID=0,     // chart's ID
                   const string name="Channel") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+-----------------------------------------------------------------------
//| Check the values of the channel's anchor points and set default values
//| for empty ones
//+-----------------------------------------------------------------------
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &tim
                              double &price2,datetime &time3,double &price
  {
//--- if the second (right) point's time is not set, it will be on the cur
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars le
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 300 points higher tha
   if(!price1)
      price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the first po
   if(!time3)
```

```
      time3=time1;
//--- if the third point's price is not set, it is equal to the second poi
   if(!price3)
      price3=price2;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the channel
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create the equidistant channel
   if(!ChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],dat
      InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpRayRight,InpHidden
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the channel's anchor points
//--- loop counter
   int h_steps=bars/6;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      //--- move the point
      if(!ChannelPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- move the first anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d1>1)
         d1-=1;
      //--- move the point
      if(!ChannelPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int v_steps=accuracy/10;
//--- move the third anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p3>1)
         p3-=1;
      //--- move the point
      if(!ChannelPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   ChannelDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_STDDEVCHANNEL

Standard Deviation Channel.



Note

For Standard Deviation Channel, it is possible to specify the mode of continuation of its display to the right (OBJPROP_RAY_RIGHT property). The mode of filling the channel with color can also be set.

OBJPROP_DEVIATION property is used to change the value of the channel deviation.

Example

The following script creates and moves Standard Deviation Channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

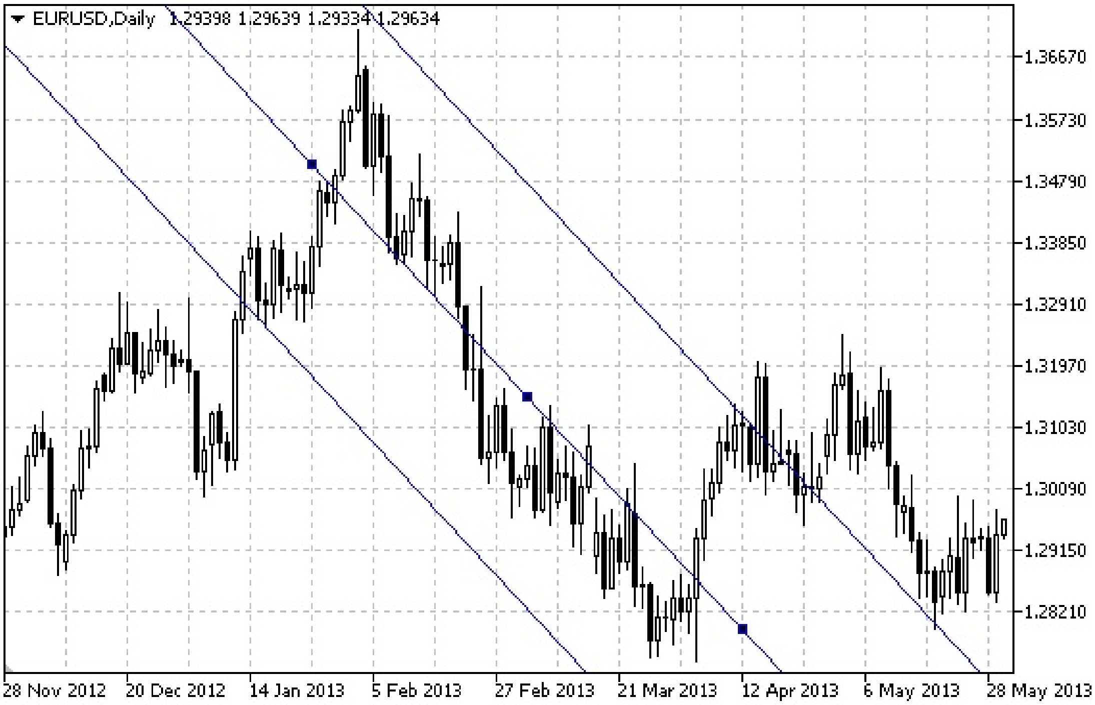
```
#property strict //--- description
#property description "Script draws \"Standard Deviation Channel\" graphic
#property description "Anchor point coordinates are set in percentage of t
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
```

```
//--- input parameters of the script
input string          InpName="StdDevChannel";     // Channel name
input int             InpDate1=10;                 // 1 st point's date, %
input int             InpDate2=40;                 // 2 nd point's date, %
input double          InpDeviation=1.0;            // Deviation
input color           InpColor=clrRed;             // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of channel lines
input int             InpWidth=1;                  // Width of channel lines
input bool            InpFill=false;               // Filling the channel wi
input bool            InpBack=false;               // Background channel
input bool            InpSelection=true;           // Highlight to move
input bool            InpRayRight=false;           // Channel's continuatio
input bool            InpHidden=true;              // Hidden in the object l
input long            InpZOrder=0;                 // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create standard deviation channel by the given coordinates       |
//+------------------------------------------------------------------+
bool StdDevChannelCreate(const long                chart_ID=0,        // chart
                         const string              name="Channel",    // chann
                         const int                 sub_window=0,      // subwi
                         datetime                  time1=0,           // first
                         datetime                  time2=0,           // secon
                         const double              deviation=1.0,     // devia
                         const color               clr=clrRed,        // chann
                         const ENUM_LINE_STYLE style=STYLE_SOLID, // style
                         const int                 width=1,           // width
                         const bool                fill=false,        // filli
                         const bool                back=false,        // in th
                         const bool                selection=true,    // highl
                         const bool                ray_right=false,   // chann
                         const bool                hidden=true,       // hidde
                         const long                z_order=0)         // prior
  {
//--- set anchor points' coordinates if they are not set
   ChangeChannelEmptyPoints(time1,time2);
//--- reset the error value
   ResetLastError();
//--- create a channel by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_STDDEVCHANNEL,sub_window,time1,0,tim
     {
      Print(__FUNCTION__,
            ": failed to create standard deviation channel! Error code = "
      return(false);
     }
//--- set deviation value affecting the channel width
   ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation);
//--- set channel color
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channe
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the channel's anchor point                                  |
//+------------------------------------------------------------------+
bool StdDevChannelPointChange(const long   chart_ID=0,     // chart's ID
                              const string name="Channel", // channel name
                              const int    point_index=0,  // anchor point
                              datetime     time=0)         // anchor point
  {
//--- if point time is not set, move the point to the current bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change the channel's deviation                                   |
//+------------------------------------------------------------------+
bool StdDevChannelDeviationChange(const long   chart_ID=0,     // chart's
                                  const string name="Channel", // channel
                                  const double deviation=1.0)  // deviatic
  {
//--- reset the error value
   ResetLastError();
//--- change trend line's slope angle
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation))
     {
      Print(__FUNCTION__,
            ": failed to change channel deviation! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the channel                                               |
//+------------------------------------------------------------------+
bool StdDevChannelDelete(const long   chart_ID=0,     // chart's ID
                         const string name="Channel") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+--------------------------------------------------------------------
//| Check the values of the channel's anchor points and set default values
//| for empty ones
//+--------------------------------------------------------------------
void ChangeChannelEmptyPoints(datetime &time1,datetime &time2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the channel
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
//--- create standard deviation channel
   if(!StdDevChannelCreate(0,InpName,0,date[d1],date[d2],InpDeviation,InpC
      InpWidth,InpFill,InpBack,InpSelection,InpRayRight,InpHidden,InpZOrde
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the channel horizontally to the right and expand it
//--- loop counter
   int h_steps=bars/2;
//--- move the channel
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following values
      if(d1<bars-1)
         d1+=1;
      if(d2<bars-1)
         d2+=1;
      //--- move the anchor points
      if(!StdDevChannelPointChange(0,InpName,0,date[d1]))
         return;
      if(!StdDevChannelPointChange(0,InpName,1,date[d2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   double v_steps=InpDeviation*2;
//--- expand the channel
   for(double i=InpDeviation;i<v_steps;i+=10.0/accuracy)
     {
      if(!StdDevChannelDeviationChange(0,InpName,i))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   StdDevChannelDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_REGRESSION

Linear Regression Channel.



Note

For Linear Regression Channel, it is possible to specify the mode of continuation of its display to the right (OBJPROP_RAY_RIGHT property). The mode of filling the channel with color can also be set.

Example

The following script creates and moves Linear Regression Channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

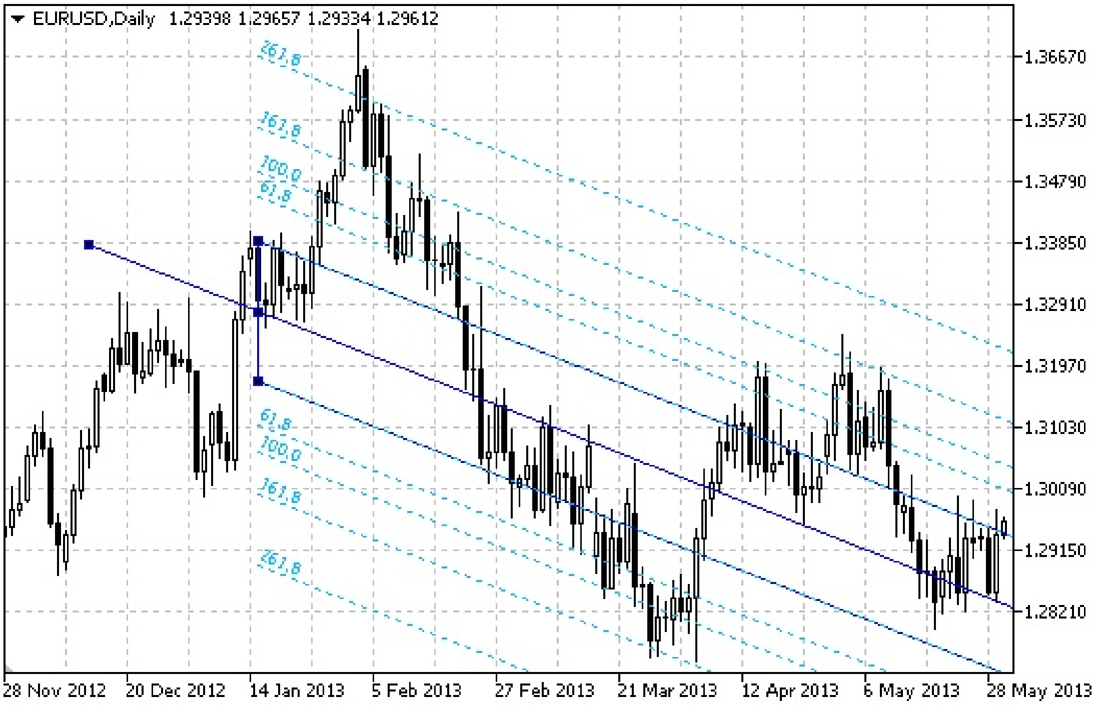
```
#property strict //--- description
#property description "Script draws \"Linear Regression Channel\" graphica
#property description "Anchor point coordinates are set in percentage of t
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Regression"; // Channel name
input int             InpDate1=10;          // 1 st point's date, %
input int             InpDate2=40;          // 2 nd point's date, %
input color           InpColor=clrRed;      // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH;  // Style of channel lines
input int             InpWidth=1;           // Width of channel lines
input bool            InpFill=false;        // Filling the channel with co
input bool            InpBack=false;        // Background channel
input bool            InpSelection=true;    // Highlight to move
input bool            InpRayRight=false;    // Channel's continuation to t
input bool            InpHidden=true;       // Hidden in the object list
input long            InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Linear Regression Channel by the given coordinates        |
//+------------------------------------------------------------------+
bool RegressionCreate(const long            chart_ID=0,        // chart's
                      const string          name="Regression", // channel
                      const int             sub_window=0,      // subwindo
                      datetime              time1=0,           // first po
                      datetime              time2=0,           // second p
                      const color           clr=clrRed,        // channel
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // style of
                      const int             width=1,           // width of
                      const bool            fill=false,        // filling
                      const bool            back=false,        // in the b
                      const bool            selection=true,    // highligh
                      const bool            ray_right=false,   // channel'
                      const bool            hidden=true,       // hidden i
                      const long            z_order=0)         // priority
  {
//--- set anchor points' coordinates if they are not set
   ChangeRegressionEmptyPoints(time1,time2);
//--- reset the error value
   ResetLastError();
//--- create a channel by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_REGRESSION,sub_window,time1,0,time2,
     {
      Print(__FUNCTION__,
            ": failed to create linear regression channel! Error code = ",
      return(false);
     }
//--- set channel color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

```
//--- enable (true) or disable (false) the mode of highlighting the channe
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the channel's anchor point                                  |
//+------------------------------------------------------------------+
bool RegressionPointChange(const long   chart_ID=0,     // chart's ID
                           const string name="Channel", // channel name
                           const int    point_index=0,  // anchor point in
                           datetime     time=0)         // anchor point ti
  {
//--- if point time is not set, move the point to the current bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the channel                                               |
//+------------------------------------------------------------------+
bool RegressionDelete(const long   chart_ID=0,     // chart's ID
                      const string name="Channel") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
```

```
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the channel! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------------
//| Check the values of the channel's anchor points and set default values
//| for empty ones
//+------------------------------------------------------------------------
void ChangeRegressionEmptyPoints(datetime &time1,datetime &time2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the channel
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
//--- create linear regression channel
   if(!RegressionCreate(0,InpName,0,date[d1],date[d2],InpColor,InpStyle,Ir
      InpFill,InpBack,InpSelection,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, the channel horizontally to the right
//--- loop counter
   int h_steps=bars/2;
//--- move the channel
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following values
      if(d1<bars-1)
         d1+=1;
      if(d2<bars-1)
         d2+=1;
      //--- move the anchor points
      if(!RegressionPointChange(0,InpName,0,date[d1]))
         return;
      if(!RegressionPointChange(0,InpName,1,date[d2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   RegressionDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_PITCHFORK

Andrews Pitchfork.



Note

For Andrews Pitchfork, it is possible to specify the mode of continuation of its display to the right (OBJPROP_RAY_RIGHT property).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Andrews Pitchfork on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

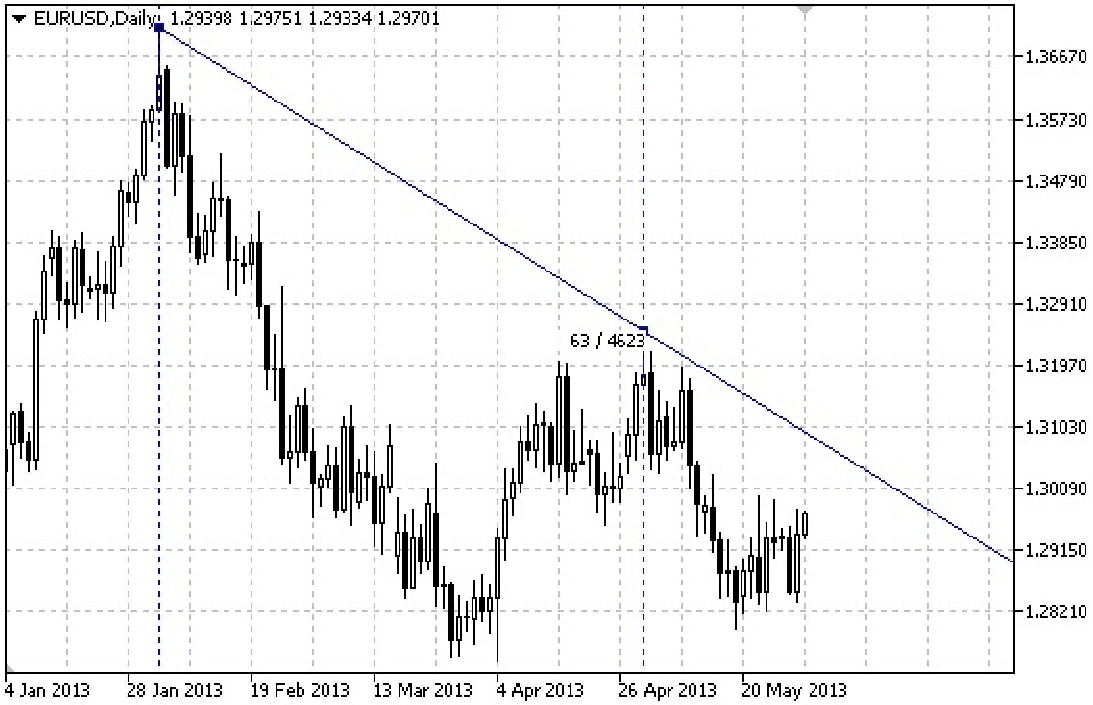
```
#property strict //--- description
#property description "Script draws \"Andrews Pitchfork\" graphical object
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Pitchfork";   // Pitchfork name
input int             InpDate1=14;           // 1 st point's date, %
input int             InpPrice1=40;          // 1 st point's price, %
input int             InpDate2=18;           // 2 nd point's date, %
input int             InpPrice2=50;          // 2 nd point's price, %
input int                InpDate3=18;          // 3 rd point's date, %
input int                InpPrice3=30;         // 3 rd point's price, %
input color              InpColor=clrRed;      // Pitchfork color
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID; // Style of pitchfork lines
input int                InpWidth=1;           // Width of pitchfork lines
input bool               InpBack=false;        // Background pitchfork
input bool               InpSelection=true;    // Highlight to move
input bool               InpRayRight=false;    // Pitchfork's continuation to
input bool               InpHidden=true;       // Hidden in the object list
input long               InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Andrews' Pitchfork by the given coordinates               |
//+------------------------------------------------------------------+
bool PitchforkCreate(const long            chart_ID=0,        // chart's I
                     const string          name="Pitchfork",  // pitchfork
                     const int             sub_window=0,      // subwindow
                     datetime              time1=0,           // first poi
                     double                price1=0,          // first poi
                     datetime              time2=0,           // second po
                     double                price2=0,          // second po
                     datetime              time3=0,           // third poi
                     double                price3=0,          // third poi
                     const color           clr=clrRed,        // color of
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // style of
                     const int             width=1,           // width of
                     const bool            back=false,        // in the ba
                     const bool            selection=true,    // highlight
                     const bool            ray_right=false,   // pitchfork
                     const bool            hidden=true,       // hidden in
                     const long            z_order=0)         // priority
  {
//--- set anchor points' coordinates if they are not set
   ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- create Andrews' Pitchfork by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_PITCHFORK,sub_window,time1,price1,ti
     {
      Print(__FUNCTION__,
            ": failed to create \"Andrews' Pitchfork\"! Error code = ",Get
      return(false);
     }
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the pitchf
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the pit
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of Andrews' Pitchfork levels and their parameters     |
//+------------------------------------------------------------------+
bool PitchforkLevelsSet(int             levels,           // number of lev
                        double          &values[],        // values of lev
                        color           &colors[],        // color of leve
                        ENUM_LINE_STYLE &styles[],        // style of leve
                        int             &widths[],        // width of leve
                        const long      chart_ID=0,       // chart's ID
                        const string    name="Pitchfork") // pitchfork nam
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Andrews' Pitchfork anchor point                             |
//+------------------------------------------------------------------+
bool PitchforkPointChange(const long   chart_ID=0,        // chart's ID
                          const string name="Pitchfork", // channel name
                          const int    point_index=0,     // anchor point i
                          datetime     time=0,            // anchor point t
                          double       price=0)           // anchor point p
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Andrews Pitchfork                                        |
//+------------------------------------------------------------------+
bool PitchforkDelete(const long   chart_ID=0,        // chart's ID
                     const string name="Pitchfork") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Andrews' Pitchfork\"! Error code = ",Get
      return(false);
     }
//--- successful execution
   return(true);
  }
//+-----------------------------------------------------------------------+
//| Check the values of Andrews' Pitchfork anchor points and set default |
//| values for empty ones                                                 |
//+-----------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &tim
                              double &price2,datetime &time3,double &price
  {
//--- if the second (upper right) point's time is not set, it will be on t
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars le
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 200 points below the
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the second p
   if(!time3)
      time3=time2;
//--- if the third point's price is not set, move it 200 points lower than
   if(!price3)
      price3=price1-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Andrews' Pitchfork anchc
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Andrews' Pitchfork
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create the pitchfork
   if(!PitchforkCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],c
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayRight,InpHidde
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the pitchfork's anchor points
//--- loop counter
```

```
   int v_steps=accuracy/10;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!PitchforkPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars/8;
//--- move the third anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d3<bars-1)
         d3+=1;
      //--- move the point
      if(!PitchforkPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy/10;
//--- move the second anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2>1)
         p2-=1;
      //--- move the point
      if(!PitchforkPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the pitchfork from the chart
   PitchforkDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_GANNLINE

Gann Line.



Note

For Gann Line, it is possible to specify the mode of continuation of its display to the right (OBJPROP_RAY_RIGHT property).

Both Gann angle with a scale and coordinates of the second anchor point can be used to set the slope of the line.

Example

The following script creates and moves Gann Line on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Gann Line\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="GannLine";        // Line name
input int             InpDate1=20;               // 1 st point's date, %
input int               InpPrice1=75;            // 1 st point's price, %
input int               InpDate2=80;             // 2 nd point's date, %
input double            InpAngle=0.0;            // Gann Angle
input double            InpScale=1.0;            // Scale
input color             InpColor=clrRed;         // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int               InpWidth=1;              // Line width
input bool              InpBack=false;           // Background line
input bool              InpSelection=true;       // Highlight to move
input bool              InpRayRight=true;        // Line's continuation to
input bool              InpHidden=true;          // Hidden in the object l
input long              InpZOrder=0;             // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Gann Line by the coordinates, angle and scale             |
//+------------------------------------------------------------------+
bool GannLineCreate(const long            chart_ID=0,        // chart's ID
                    const string          name="GannLine",   // line name
                    const int             sub_window=0,      // subwindow
                    datetime              time1=0,           // first poin
                    double                price1=0,          // first poin
                    datetime              time2=0,           // second poi
                    const double          angle=1.0,         // Gann angle
                    const double          scale=1.0,         // scale
                    const color           clr=clrRed,        // line color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                    const int             width=1,           // line width
                    const bool            back=false,        // in the bac
                    const bool            selection=true,    // highlight
                    const bool            ray_right=true,    // line's con
                    const bool            hidden=true,       // hidden in
                    const long            z_order=0)         // priority f
  {
//--- set anchor points' coordinates if they are not set
   ChangeGannLineEmptyPoints(time1,price1,time2);
//--- reset the error value
   ResetLastError();
//--- create Gann Line by the given coordinates
//--- correct coordinate of the second anchor point is redefined
//--- automatically after Gann angle and/or the scale changes,
   if(!ObjectCreate(chart_ID,name,OBJ_GANNLINE,sub_window,time1,price1,tim
     {
      Print(__FUNCTION__,
            ": failed to create \"Gann Line\"! Error code = ",GetLastError
      return(false);
     }
//--- change Gann angle
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
```

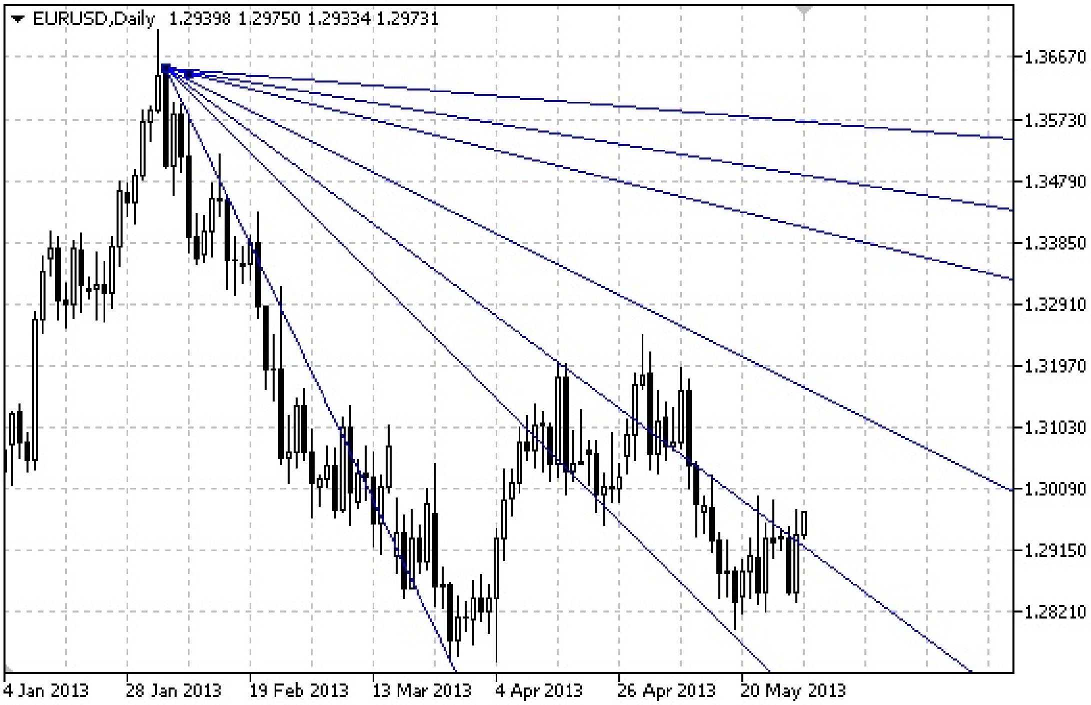
```
//--- change the scale (number of pips per bar)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- set line color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line display style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the lines
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the lin
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Gann Line anchor point                                      |
//+------------------------------------------------------------------+
bool GannLinePointChange(const long   chart_ID=0,      // chart's ID
                         const string name="GannLine", // line name
                         const int    point_index=0,   // anchor point ind
                         datetime     time=0,          // anchor point tim
                         double       price=0)         // anchor point pri
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the line's anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann angle                                                |
//+------------------------------------------------------------------+
bool GannLineAngleChange(const long   chart_ID=0,       // chart's ID
                         const string name="GannLine", // line name
                         const double angle=1.0)       // Gann angle
  {
//--- reset the error value
   ResetLastError();
//--- change Gann angle
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
     {
      Print(__FUNCTION__,
            ": failed to change Gann angle! Error code = ",GetLastError())
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Line's scale                                         |
//+------------------------------------------------------------------+
bool GannLineScaleChange(const long   chart_ID=0,       // chart's ID
                         const string name="GannLine", // line name
                         const double scale=1.0)       // scale
  {
//--- reset the error value
   ResetLastError();
//--- change the scale (number of pips per bar)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
      Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function removes Gann Line from the chart                   |
//+------------------------------------------------------------------+
bool GannLineDelete(const long   chart_ID=0,       // chart's ID
                    const string name="GannLine") // line name
  {
//--- reset the error value
   ResetLastError();
//--- delete Gann line
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Gann Line\"! Error code = ",GetLastError
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Gann Line anchor points and set default      |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeGannLineEmptyPoints(datetime &time1,double &price1,datetime &ti
  {
//--- if the second point's time is not set, it will be on the current ba
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left fro
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
```

```
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing line anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Gann Line
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- create Gann Line
   if(!GannLineCreate(0,InpName,0,date[d1],price[p1],date[d2],InpAngle,Inp
      InpStyle,InpWidth,InpBack,InpSelection,InpRayRight,InpHidden,InpZOr
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the line's anchor point and change the angle
//--- loop counter
   int v_steps=accuracy/2;
//--- move the first anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
```

```
         p1-=1;
      //--- move the point
      if(!GannLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- half a second of delay
   Sleep(500);
//--- define the current value of Gann angle (changed
//--- after moving the first anchor point)
   double curr_angle;
   if(!ObjectGetDouble(0,InpName,OBJPROP_ANGLE,0,curr_angle))
      return;
//--- loop counter
   v_steps=accuracy/8;
//--- change Gann angle
   for(int i=0;i<v_steps;i++)
     {
      if(!GannLineAngleChange(0,InpName,curr_angle-0.05*i))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the line from the chart
   GannLineDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_GANNFAN

Gann Fan.



Note

For Gann Fan, it is possible to specify trend type from ENUM_GANN_DIRECTION enumeration. By adjusting the scale value (OBJPROP_SCALE), it is possible to change slope angle of the fan lines.

Example

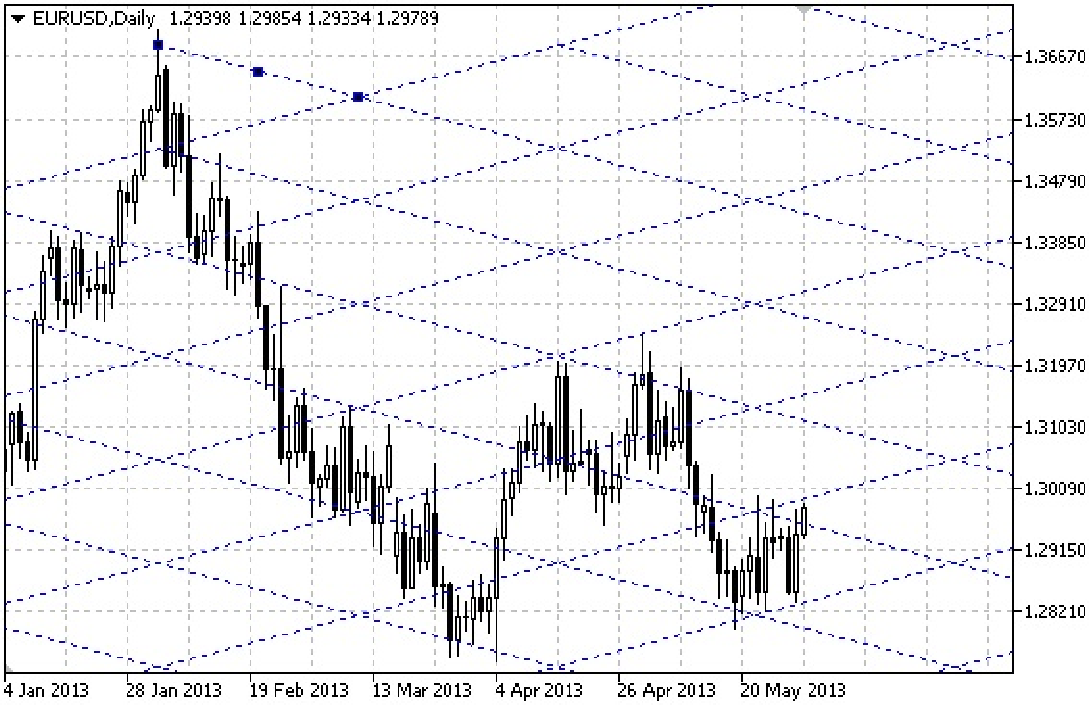
The following script creates and moves Gann Fan on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Gann Fan\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="GannFan";           // Fan name
input int             InpDate1=15;                 // 1 st point's date, %
input int             InpPrice1=25;                // 1 st point's price, %
input int             InpDate2=85;               // 2 nd point's date, %
input double          InpScale=2.0;              // Scale
input bool            InpDirection=false;        // Trend direction
input color           InpColor=clrRed;           // Fan color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of fan lines
input int             InpWidth=1;                // Width of fan lines
input bool            InpBack=false;             // Background fan
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object l
input long            InpZOrder=0;               // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Gann Fan                                                  |
//+------------------------------------------------------------------+
bool GannFanCreate(const long            chart_ID=0,        // chart's ID
                   const string          name="GannFan",    // fan name
                   const int             sub_window=0,      // subwindow i
                   datetime              time1=0,           // first point
                   double                price1=0,          // first point
                   datetime              time2=0,           // second poin
                   const double          scale=1.0,         // scale
                   const bool            direction=true,    // trend direc
                   const color           clr=clrRed,        // fan color
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // style of fa
                   const int             width=1,           // width of fa
                   const bool            back=false,        // in the back
                   const bool            selection=true,    // highlight t
                   const bool            hidden=true,       // hidden in t
                   const long            z_order=0)         // priority fo
  {
//--- set anchor points' coordinates if they are not set
   ChangeGannFanEmptyPoints(time1,price1,time2);
//--- reset the error value
   ResetLastError();
//--- create Gann Fan by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_GANNFAN,sub_window,time1,price1,time
     {
      Print(__FUNCTION__,
            ": failed to create \"Gann Fan\"! Error code = ",GetLastError(
      return(false);
     }
//--- change the scale (number of pips per bar)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- change Gann Fan's trend direction (true - descending, false - ascend
   ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
//--- set fan color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set display style of the fan lines
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the fan lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the fan fo
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Gann Fan anchor point                                       |
//+------------------------------------------------------------------+
bool GannFanPointChange(const long   chart_ID=0,     // chart's ID
                        const string name="GannFan", // fan name
                        const int    point_index=0,  // anchor point index
                        datetime     time=0,         // anchor point time
                        double       price=0)        // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the fan's anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Fan's scale                                          |
//+------------------------------------------------------------------+
bool GannFanScaleChange(const long   chart_ID=0,      // chart's ID
                        const string name="GannFan", // fan name
                        const double scale=1.0)      // scale
  {
//--- reset the error value
   ResetLastError();
//--- change the scale (number of pips per bar)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
      Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Fan's trend direction                                |
//+------------------------------------------------------------------+
bool GannFanDirectionChange(const long   chart_ID=0,      // chart's ID
                            const string name="GannFan", // fan name
                            const bool   direction=true) // trend directic
  {
//--- reset the error value
   ResetLastError();
//--- change Gann Fan's trend direction
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
     {
      Print(__FUNCTION__,
            ": failed to change trend direction! Error code = ",GetLastErr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function removes Gann Fan from the chart                     |
//+------------------------------------------------------------------+
bool GannFanDelete(const long   chart_ID=0,      // chart's ID
                   const string name="GannFan") // fan name
  {
//--- reset the error value
   ResetLastError();
//--- delete Gann Fan
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Gann Fan\"! Error code = ",GetLastError(
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Gann Fan anchor points and set default        |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeGannFanEmptyPoints(datetime &time1,double &price1,datetime &tim
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of fan's anchor points
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Gann Fan
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- create Gann Fan
   if(!GannFanCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpD
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder)
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the fan's anchor point
//--- loop counter
   int v_steps=accuracy/2;
//--- move the first anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1<accuracy-1)
         p1+=1;
      //--- move the point
      if(!GannFanPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- change fan's trend direction to descending one
   GannFanDirectionChange(0,InpName,true);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the fan from the chart
   GannFanDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_GANNGRID

Gann Grid.



Note

For Gann Grid, it is possible to specify trend type from ENUM_GANN_DIRECTION. By adjusting the scale value (OBJPROP_SCALE), it is possible to change slope angle of the grid lines.

Example

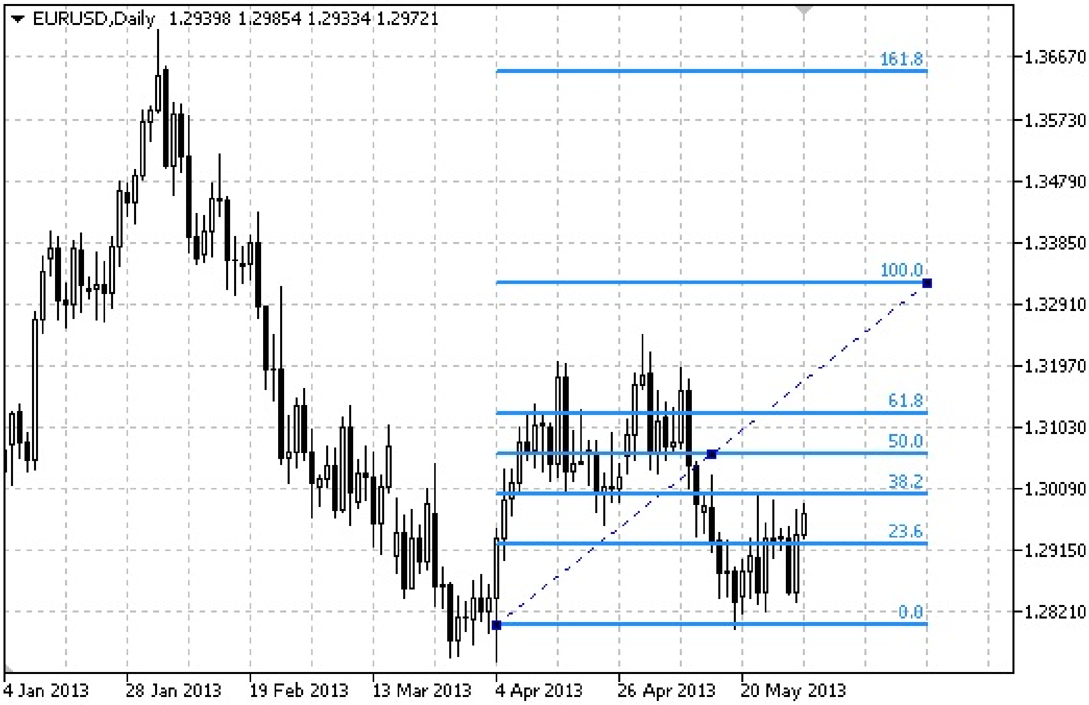
The following script creates and moves Gann Grid on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Gann Grid\" graphical object."
#property description "Anchor point coordinates of the grid are set in per
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="GannGrid";        // Grid name
input int             InpDate1=15;               // 1 st point's date, %
input int             InpPrice1=25;              // 1 st point's price, %
input int              InpDate2=35;                // 2 nd point's date, %
input double           InpScale=3.0;               // Scale
input bool             InpDirection=false;         // Trend direction
input color            InpColor=clrRed;            // Grid color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of grid lines
input int              InpWidth=1;                 // Width of fan lines
input bool             InpBack=false;              // Background grid
input bool             InpSelection=true;          // Highlight to move
input bool             InpHidden=true;             // Hidden in the object l
input long             InpZOrder=0;                // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Gann Grid                                                 |
//+------------------------------------------------------------------+
bool GannGridCreate(const long            chart_ID=0,        // chart's ID
                    const string          name="GannGrid",   // grid name
                    const int             sub_window=0,      // subwindow
                    datetime              time1=0,           // first poin
                    double                price1=0,          // first poin
                    datetime              time2=0,           // second poi
                    const double          scale=1.0,         // scale
                    const bool            direction=true,    // trend dire
                    const color           clr=clrRed,        // grid color
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // style of g
                    const int             width=1,           // width of g
                    const bool            back=false,        // in the bac
                    const bool            selection=true,    // highlight
                    const bool            hidden=true,       // hidden in
                    const long            z_order=0)         // priority f
  {
//--- set anchor points' coordinates if they are not set
   ChangeGannGridEmptyPoints(time1,price1,time2);
//--- reset the error value
   ResetLastError();
//--- create Gann Grid by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_GANNGRID,sub_window,time1,price1,tim
     {
      Print(__FUNCTION__,
            ": failed to create \"Gann Grid\"! Error code = ",GetLastError
      return(false);
     }
//--- change the scale (number of pips per bar)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- change Gann Fan's trend direction (true - descending, false - ascend
   ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
//--- set grid color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set display style of the grid lines
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the grid lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the grid f
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Gann Grid anchor point                                      |
//+------------------------------------------------------------------+
bool GannGridPointChange(const long   chart_ID=0,      // chart's ID
                         const string name="GannGrid", // grid name
                         const int    point_index=0,   // anchor point ind
                         datetime     time=0,          // anchor point tim
                         double       price=0)         // anchor point pri
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the grid's anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Grid's scale                                         |
//+------------------------------------------------------------------+
bool GannGridScaleChange(const long   chart_ID=0,       // chart's ID
                         const string name="GannGrid", // grids
                         const double scale=1.0)       // scale
  {
//--- reset the error value
   ResetLastError();
//--- change the scale (number of pips per bar)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
      Print(__FUNCTION__,
            ": failed to change the scale! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Gann Grid's trend direction                               |
//+------------------------------------------------------------------+
bool GannGridDirectionChange(const long   chart_ID=0,       // chart's ID
                             const string name="GannGrid", // grid name
                             const bool   direction=true)  // trend direct
  {
//--- reset the error value
   ResetLastError();
//--- change Gann Grid's trend direction
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
     {
      Print(__FUNCTION__,
            ": failed to change trend direction! Error code = ",GetLastErr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| The function removes Gann Fan from the chart                     |
//+------------------------------------------------------------------+
bool GannGridDelete(const long   chart_ID=0,       // chart's ID
                    const string name="GannGrid") // grid name
  {
//--- reset the error value
   ResetLastError();
//--- delete Gann Grid
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Gann Grid\"! Error code = ",GetLastError
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Gann Grid anchor points and set default      |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeGannGridEmptyPoints(datetime &time1,double &price1,datetime &ti
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the first point's time is not set, it is located 9 bars left from
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing grid anchor points' coordinates
   datetime date[];
   double    price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Gann Grid
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- create Gann Grid
   if(!GannGridCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,Inp
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder)
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the grid's anchor points
//--- loop counter
   int v_steps=accuracy/4;
//--- move the first anchor point vertically
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1<accuracy-1)
         p1+=1;
      if(!GannGridPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars/4;
//--- move the second anchor point horizontally
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      if(!GannGridPointChange(0,InpName,1,date[d2],0))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- change grid's trend direction to descending one
   GannGridDirectionChange(0,InpName,true);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the grid from the chart
   GannGridDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBO

Fibonacci Retracement.



Note

For Fibonacci Retracement, it is possible to specify the mode of continuation of its display to the right (OBJPROP_RAY_RIGHT property).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Retracement on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

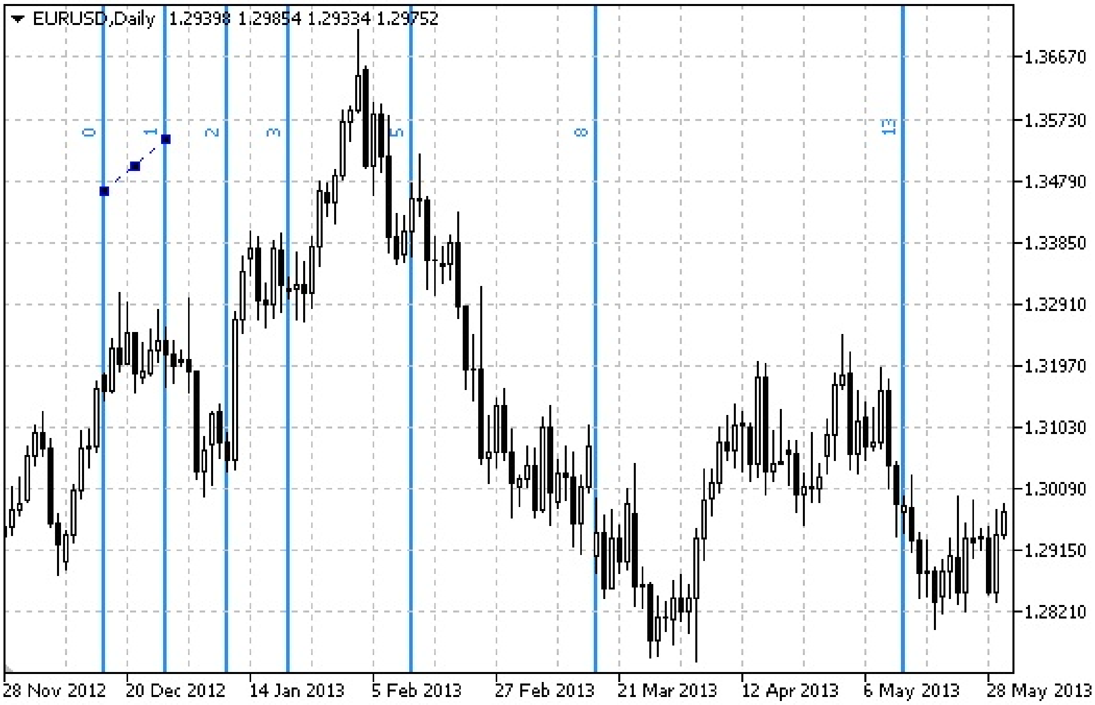
```
#property strict //--- description
#property description "Script draws \"Fibonacci Retracement\" graphical ob
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboLevels";      // Object name
input int             InpDate1=10;               // 1 st point's date, %
input int             InpPrice1=65;              // 1 st point's price, %
input int             InpDate2=90;               // 2 nd point's date, %
input int             InpPrice2=85;              // 2 nd point's price, %
input color           InpColor=clrRed;           // Object color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int             InpWidth=1;                // Line width
input bool            InpBack=false;             // Background object
input bool            InpSelection=true;         // Highlight to move
input bool            InpRayRight=false;         // Object's continuation
input bool            InpHidden=true;            // Hidden in the object l
input long            InpZOrder=0;               // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Fibonacci Retracement by the given coordinates            |
//+------------------------------------------------------------------+
bool FiboLevelsCreate(const long            chart_ID=0,        // chart's
                      const string          name="FiboLevels", // object r
                      const int             sub_window=0,      // subwindo
                      datetime              time1=0,           // first po
                      double                price1=0,          // first po
                      datetime              time2=0,           // second p
                      double                price2=0,          // second p
                      const color           clr=clrRed,        // object c
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // object l
                      const int             width=1,           // object l
                      const bool            back=false,        // in the b
                      const bool            selection=true,    // highligh
                      const bool            ray_right=false,   // object's
                      const bool            hidden=true,       // hidden i
                      const long            z_order=0)         // priority
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboLevelsEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- Create Fibonacci Retracement by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBO,sub_window,time1,price1,time2,p
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Retracement\"! Error code = ",
      return(false);
     }
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
```

```
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channe
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the obj
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboLevelsSet(int             levels,            // number of level l
                   double          &values[],         // values of level l
                   color           &colors[],         // color of level li
                   ENUM_LINE_STYLE &styles[],         // style of level li
                   int             &widths[],         // width of level li
                   const long      chart_ID=0,        // chart's ID
                   const string    name="FiboLevels") // object name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Retracement anchor point                          |
//+------------------------------------------------------------------+
bool FiboLevelsPointChange(const long   chart_ID=0,        // chart's ID
                           const string name="FiboLevels", // object name
                           const int    point_index=0,     // anchor point
                           datetime     time=0,            // anchor point
                           double       price=0)           // anchor point
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibonacci Retracement                                     |
//+------------------------------------------------------------------+
bool FiboLevelsDelete(const long   chart_ID=0,        // chart's ID
                      const string name="FiboLevels") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Retracement\"! Error code = ",
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Retracement anchor points and set  |
//| default values for empty ones                                    |
//+------------------------------------------------------------------+
void ChangeFiboLevelsEmptyPoints(datetime &time1,double &price1,
                                 datetime &time2,double &price2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first point's time is not set, it is located 9 bars left from
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 200 points below the
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
```

```
//--- for setting and changing the coordinates of Fibonacci Retracement ar
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Retracement
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
   if(!FiboLevelsCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpStyle,InpWidth,InpBack,InpSelection,InpRayRight,InpHidden,InpZOrc
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int v_steps=accuracy*2/5;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!FiboLevelsPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy*4/5;
//--- move the second anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2>1)
         p2-=1;
      //--- move the point
      if(!FiboLevelsPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   FiboLevelsDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBOTIMES

Fibonacci Time Zones.



Note

For "Fibonacci Time Zones", it is possible to specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Time Zones on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

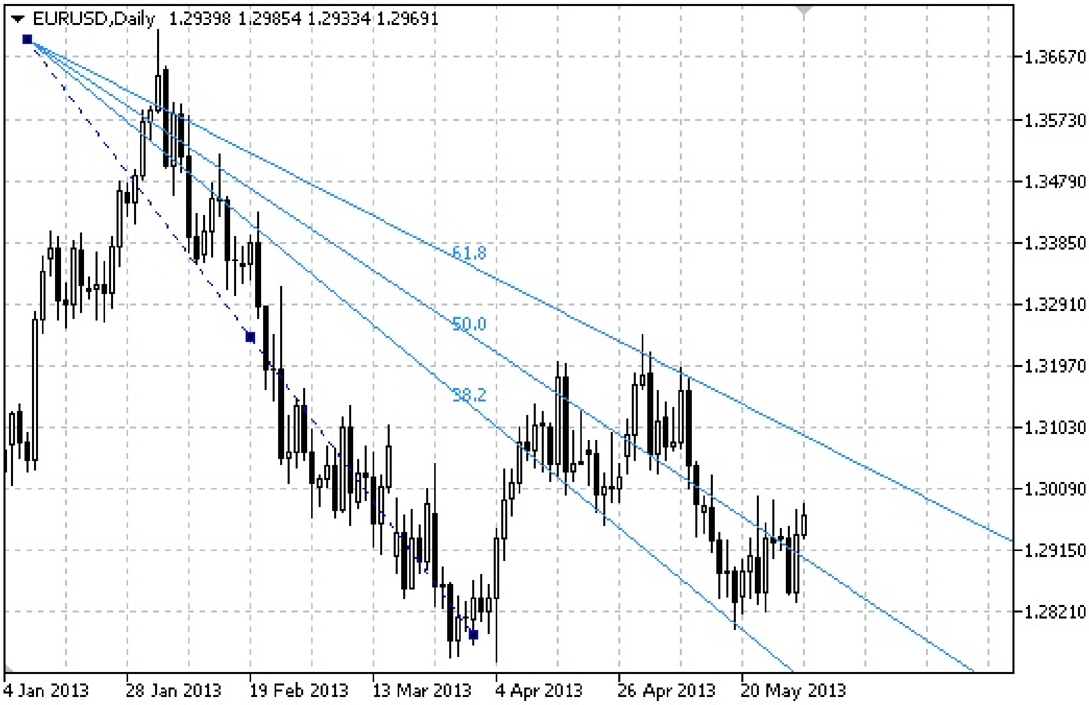
```
#property strict //--- description
#property description "Script draws \"Fibonacci Time Zones\" graphical obj
#property description "Anchor point coordinates are set in percentage of t
#property description "the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboTimes";         // Object name
input int             InpDate1=10;                 // 1 st point's date, %
input int             InpPrice1=45;                // 1 st point's price, %
input int             InpDate2=20;                 // 2 nd point's date, %
input int             InpPrice2=55;                // 2 nd point's price, %
input color           InpColor=clrRed;             // Object color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int             InpWidth=1;                // Line width
input bool            InpBack=false;             // Background object
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object l
input long            InpZOrder=0;               // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Fibonacci Time Zones by the given coordinates             |
//+------------------------------------------------------------------+
bool FiboTimesCreate(const long            chart_ID=0,        // chart's I
                     const string          name="FiboTimes",  // object na
                     const int             sub_window=0,      // subwindow
                     datetime              time1=0,           // first poi
                     double                price1=0,          // first poi
                     datetime              time2=0,           // second po
                     double                price2=0,          // second po
                     const color           clr=clrRed,        // object co
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // object li
                     const int             width=1,           // object li
                     const bool            back=false,        // in the ba
                     const bool            selection=true,    // highlight
                     const bool            hidden=true,       // hidden in
                     const long            z_order=0)         // priority
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboTimesEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create Fibonacci Time Zones by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOTIMES,sub_window,time1,price1,ti
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Time Zones\"! Error code = ",G
      return(false);
     }
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channe
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboTimesLevelsSet(int             levels,           // number of lev
                        double          &values[],        // values of lev
                        color           &colors[],        // color of leve
                        ENUM_LINE_STYLE &styles[],       // style of leve
                        int             &widths[],        // width of leve
                        const long      chart_ID=0,       // chart's ID
                        const string    name="FiboTimes") // object name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(val
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Time Zones anchor point                            |
//+------------------------------------------------------------------+
bool FiboTimesPointChange(const long   chart_ID=0,       // chart's ID
                          const string name="FiboTimes", // object name
                          const int    point_index=0,    // anchor point i
                          datetime     time=0,           // anchor point t
                          double       price=0)          // anchor point p
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibonacci Time Zones                                       |
//+------------------------------------------------------------------+
bool FiboTimesDelete(const long   chart_ID=0,       // chart's ID
                     const string name="FiboTimes") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Time Zones\"! Error code = ",G
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Time Zones and                      |
//| set default values for empty ones                                 |
//+------------------------------------------------------------------+
void ChangeFiboTimesEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 2 bars left fro
   if(!time2)
     {
      //--- array for receiving the open time of the last 3 bars
      datetime temp[3];
      CopyTime(Symbol(),Period(),time1,3,temp);
      //--- set the first point 2 bars left from the second one
      time2=temp[0];
     }
//--- if the second point's price is not set, it is equal to the first poi
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Time Zones and
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Time Zones
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
   if(!FiboTimesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int h_steps=bars*2/5;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      //--- move the point
      if(!FiboTimesPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
```

```
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   h_steps=bars*3/5;
//--- move the first anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d1<bars-1)
         d1+=1;
      //--- move the point
      if(!FiboTimesPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   FiboTimesDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBOFAN

Fibonacci Fan.



Note

For "Fibonacci Fan", it is possible to specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Fan on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

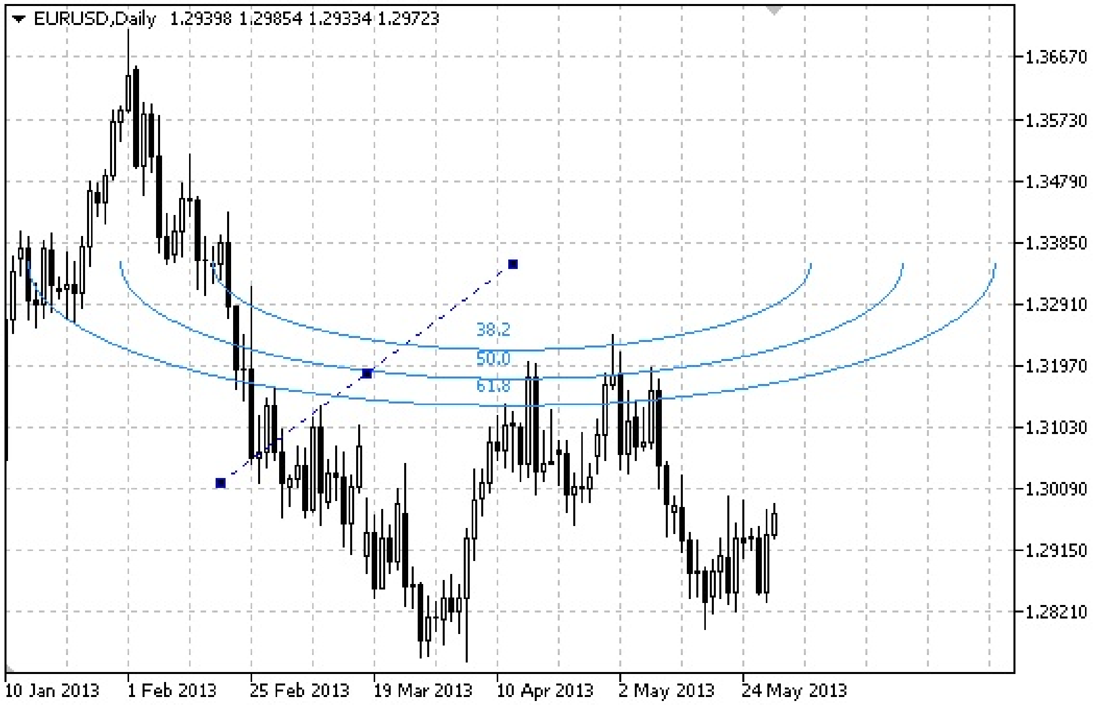
```
#property strict //--- description
#property description "Script draws \"Fibonacci Fan\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboFan";          // Fan name
input int             InpDate1=10;                // 1 st point's date, %
input int             InpPrice1=25;               // 1 st point's price, %
input int             InpDate2=30;                // 2 nd point's date, %
input int                InpPrice2=50;               // 2 nd point's price, %
input color              InpColor=clrRed;            // Fan line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int                InpWidth=1;                 // Line width
input bool               InpBack=false;              // Background object
input bool               InpSelection=true;          // Highlight to move
input bool               InpHidden=true;             // Hidden in the object l
input long               InpZOrder=0;                // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Fibonacci Fan by the given coordinates                      |
//+------------------------------------------------------------------+
bool FiboFanCreate(const long              chart_ID=0,        // chart's ID
                   const string            name="FiboFan",    // fan name
                   const int               sub_window=0,      // subwindow i
                   datetime                time1=0,           // first point
                   double                  price1=0,          // first point
                   datetime                time2=0,           // second poin
                   double                  price2=0,          // second poin
                   const color             clr=clrRed,        // fan line co
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // fan line st
                   const int               width=1,           // fan line wi
                   const bool              back=false,        // in the back
                   const bool              selection=true,    // highlight t
                   const bool              hidden=true,       // hidden in t
                   const long              z_order=0)         // priority fo
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboFanEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create Fibonacci Fan by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOFAN,sub_window,time1,price1,time
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Fan\"! Error code = ",GetLastE
      return(false);
     }
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the fan fo
//--- when creating a graphical object using ObjectCreate function, the ob
```

```
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboFanLevelsSet(int             levels,          // number of level l
                      double          &values[],       // values of level l
                      color           &colors[],       // color of level li
                      ENUM_LINE_STYLE &styles[],       // style of level li
                      int             &widths[],       // width of level li
                      const long      chart_ID=0,      // chart's ID
                      const string    name="FiboFan")  // fan name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Fan anchor point                                  |
//+------------------------------------------------------------------+
bool FiboFanPointChange(const long   chart_ID=0,     // chart's ID
                        const string name="FiboFan", // fan name
                        const int    point_index=0,  // anchor point index
                        datetime     time=0,         // anchor point time
                        double       price=0)        // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibo name                                                 |
//+------------------------------------------------------------------+
bool FiboFanDelete(const long   chart_ID=0,     // chart's ID
                   const string name="FiboFan") // fan name
  {
//--- reset the error value
   ResetLastError();
//--- delete the fan
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Fan\"! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Fan anchor points and set          |
//| default values for empty ones                                    |
//+------------------------------------------------------------------+
void ChangeFiboFanEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first point's time is not set, it is located 9 bars left from
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 200 points below the
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Fan anchor poi
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
```

```
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Fan
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
   if(!FiboFanCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder)
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the fan's anchor points
//--- loop counter
   int v_steps=accuracy/2;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1<accuracy-1)
         p1+=1;
      //--- move the point
      if(!FiboFanPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars/4;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      //--- move the point
      if(!FiboFanPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   FiboFanDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBOARC

Fibonacci Arcs.



Note

For "Fibonacci Arcs", it is possible to specify the display mode of the entire ellipse. Curvature radius can be specified by changing the scale and coordinates of the anchor points.

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Arcs on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Fibonacci Arcs\" graphical object."
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboArc";          // Object name
input int             InpDate1=25;                // 1 st point's date, %
input int             InpPrice1=25;              // 1 st point's price, %
input int             InpDate2=35;               // 2 nd point's date, %
input int             InpPrice2=55;              // 2 nd point's price, %
input double          InpScale=3.0;              // Scale
input bool            InpFullEllipse=true;       // Shape of the arcs
input color           InpColor=clrRed;           // Line color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Line style
input int             InpWidth=1;                // Line width
input bool            InpBack=false;             // Background object
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object l
input long            InpZOrder=0;               // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Fibonacci Arcs by the given coordinates                   |
//+------------------------------------------------------------------+
bool FiboArcCreate(const long            chart_ID=0,         // chart's ID
                   const string          name="FiboArc",     // object nam
                   const int             sub_window=0,       // subwindow
                   datetime              time1=0,            // first poin
                   double                price1=0,           // first poin
                   datetime              time2=0,            // second poi
                   double                price2=0,           // second poi
                   const double          scale=1.0,          // scale
                   const bool            full_ellipse=false, // shape of t
                   const color           clr=clrRed,         // line color
                   const ENUM_LINE_STYLE style=STYLE_SOLID,  // line style
                   const int             width=1,            // line width
                   const bool            back=false,         // in the bac
                   const bool            selection=true,     // highlight
                   const bool            hidden=true,        // hidden in
                   const long            z_order=0)          // priority f
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboArcEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create Fibonacci Arcs by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOARC,sub_window,time1,price1,time
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Arcs\"! Error code = ",GetLast
      return(false);
     }
//--- set the scale
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- set display of the arcs as a full ellipse (true) or a half of it (fa
   ObjectSetInteger(chart_ID,name,OBJPROP_ELLIPSE,full_ellipse);
```

```
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set line width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the arcs f
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboArcLevelsSet(int             levels,         // number of level l
                      double          &values[],      // values of level l
                      color           &colors[],      // color of level li
                      ENUM_LINE_STYLE &styles[],      // style of level li
                      int             &widths[],      // width of level li
                      const long      chart_ID=0,     // chart's ID
                      const string    name="FiboArc") // object name
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Arcs anchor point                                 |
//+------------------------------------------------------------------+
bool FiboArcPointChange(const long   chart_ID=0,     // chart's ID
                        const string name="FiboArc", // object name
                        const int    point_index=0,  // anchor point index
                        datetime     time=0,         // anchor point time
                        double       price=0)        // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibonacci Arcs                                            |
//+------------------------------------------------------------------+
bool FiboArcDelete(const long   chart_ID=0,     // chart's ID
                   const string name="FiboArc") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
```

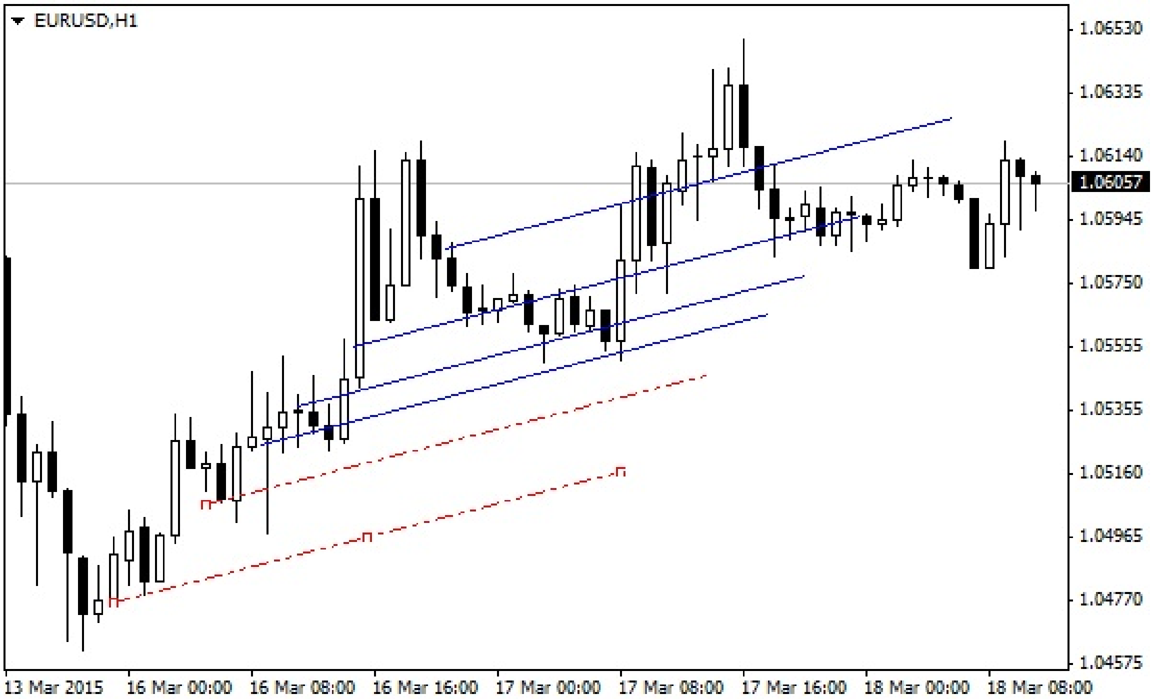
```
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Arcs\"! Error code = ",GetLast
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Arcs anchor points and set default |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeFiboArcEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
  {
//--- if the second point's time is not set, it will be on the current bar
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first point's time is not set, it is located 9 bars left from
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 300 points below the
   if(!price1)
      price1=price2-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
```

```
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing the coordinates of Fibonacci Arcs anchor po
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Arcs
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an object
   if(!FiboArcCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],Inp
      InpFullEllipse,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHi
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int v_steps=accuracy/5;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1<accuracy-1)
```

```
         p1+=1;
      //--- move the point
      if(!FiboArcPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars/5;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2<bars-1)
         d2+=1;
      //--- move the point
      if(!FiboArcPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   FiboArcDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_FIBOCHANNEL

Fibonacci Channel.



Note

For Fibonacci Channel, it is possible to specify the mode of continuation of its display to the chart (OBJPROP_RAY property).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Channel on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

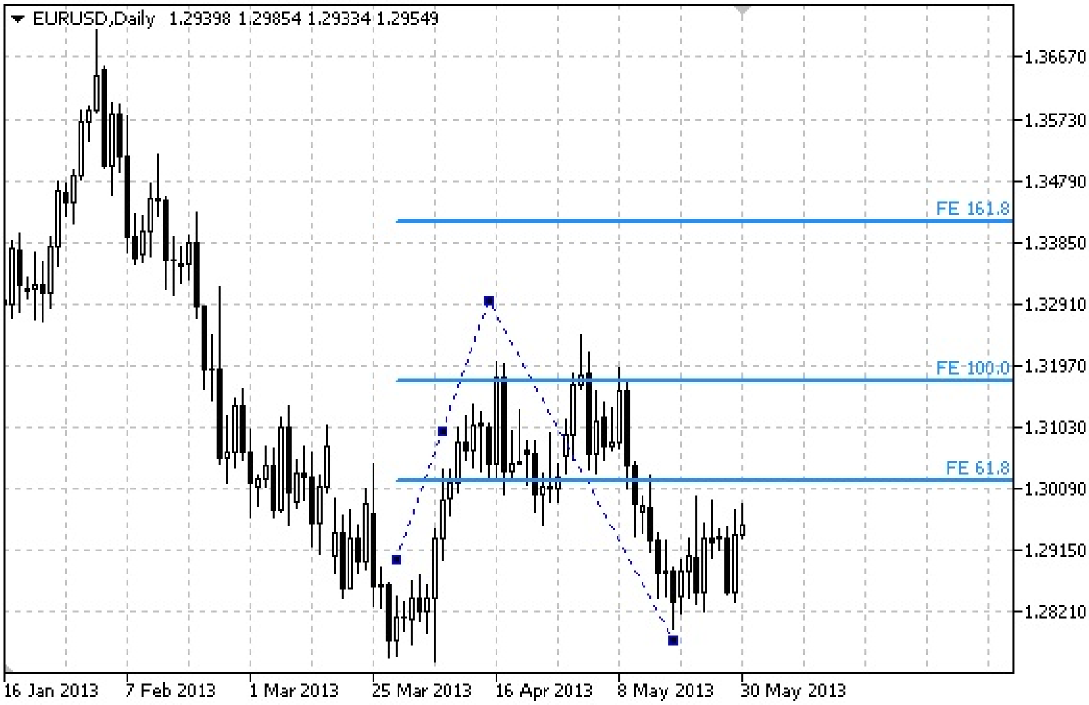
```
#property strict //--- description
#property description "Script draws \"Fibonacci Channel\" graphical object
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboChannel";      // Channel name
input int             InpDate1=20;                // 1 st point's date, %
input int                   InpPrice1=10;                 // 1 st point's price, %
input int                   InpDate2=60;                  // 2 nd point's date, %
input int                   InpPrice2=30;                 // 2 nd point's price, %
input int                   InpDate3=20;                  // 3 rd point's date, %
input int                   InpPrice3=25;                 // 3 rd point's price, %
input color                 InpColor=clrRed;              // Channel color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of channel lines
input int                   InpWidth=1;                   // Width of channel lines
input bool                  InpBack=false;                // Background channel
input bool                  InpSelection=true;            // Highlight to move
input bool                  InpRay=false;                 // Channel's continuation
input bool                  InpHidden=true;               // Hidden in the object l
input long                  InpZOrder=0;                  // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Fibonacci Channel by the given coordinates                |
//+------------------------------------------------------------------+
bool FiboChannelCreate(const long            chart_ID=0,          // chart'
                       const string          name="FiboChannel", // channe
                       const int             sub_window=0,        // subwin
                       datetime              time1=0,             // first
                       double                price1=0,            // first
                       datetime              time2=0,             // second
                       double                price2=0,            // second
                       datetime              time3=0,             // third
                       double                price3=0,            // third
                       const color           clr=clrRed,          // channe
                       const ENUM_LINE_STYLE style=STYLE_SOLID,   // style
                       const int             width=1,             // width
                       const bool            back=false,          // in the
                       const bool            selection=true,      // highli
                       const bool            ray=false,           // channe
                       const bool            hidden=true,         // hidden
                       const long            z_order=0)           // priori
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- create a channel by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOCHANNEL,sub_window,time1,price1,
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Channel\"! Error code = ",GetI
      return(false);
     }
//--- set channel color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
```

```
//--- set style of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the channel lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channe
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY,ray);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboChannelLevelsSet(int             levels,             // number of
                          double          &values[],          // values of
                          color           &colors[],          // color of
                          ENUM_LINE_STYLE &styles[],          // style of
                          int             &widths[],          // width of
                          const long      chart_ID=0,         // chart's I
                          const string    name="FiboChannel") // object na
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Channel anchor point                              |
//+------------------------------------------------------------------+
bool FiboChannelPointChange(const long   chart_ID=0,         // chart's ID
                            const string name="FiboChannel", // channel na
                            const int    point_index=0,      // anchor poi
                            datetime     time=0,             // anchor poi
                            double       price=0)            // anchor poi
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the channel                                               |
//+------------------------------------------------------------------+
bool FiboChannelDelete(const long   chart_ID=0,         // chart's ID
                       const string name="FiboChannel") // channel name
  {
//--- reset the error value
   ResetLastError();
//--- delete the channel
   if(!ObjectDelete(chart_ID,name))
```

```
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Channel\"! Error code = ",GetI
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Channel anchor points and set      |
//| default values for empty ones                                    |
//+------------------------------------------------------------------+
void ChangeFiboChannelEmptyPoints(datetime &time1,double &price1,datetime
                                  double &price2,datetime &time3,double &p
  {
//--- if the second (right) point's time is not set, it will be on the cur
   if(!time2)
      time2=TimeCurrent();
//--- if the second point's price is not set, it will have Bid value
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars le
   if(!time1)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, move it 300 points higher tha
   if(!price1)
      price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the first po
   if(!time3)
      time3=time1;
//--- if the third point's price is not set, it is equal to the second poi
   if(!price3)
      price3=price2;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing channel anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the channel
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create Fibonacci Channel
   if(!FiboChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2]
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRay,InpHidden,Inp
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the channel's anchor points
//--- loop counter
   int h_steps=bars/10;
//--- move the first anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d1>1)
         d1-=1;
      //--- move the point
      if(!FiboChannelPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int v_steps=accuracy/10;
//--- move the second anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2>1)
         p2-=1;
      //--- move the point
      if(!FiboChannelPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy/15;
//--- move the third anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p3<accuracy-1)
         p3+=1;
      //--- move the point
      if(!FiboChannelPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   FiboChannelDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_EXPANSION

Fibonacci Expansion.



Note

For "Fibonacci Expansion", it is possible to specify the mode of continuation of its display to the right (OBJPROP_RAY_RIGHT property).

You can also specify the number of line-levels, their values and color.

Example

The following script creates and moves Fibonacci Expansion on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Fibonacci Expansion\"graphical objec
#property description "Anchor point coordinates are set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="FiboExpansion";    // Object name
input int             InpDate1=10;                // 1 st point's date, %
input int             InpPrice1=55;              // 1 st point's price, %
input int             InpDate2=30;               // 2 nd point's date, %
input int             InpPrice2=10;              // 2 nd point's price, %
input int             InpDate3=80;               // 3 rd point's date, %
input int             InpPrice3=75;              // 3 rd point's price, %
input color           InpColor=clrRed;           // Object color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of lines
input int             InpWidth=1;                // Width of the lines
input bool            InpBack=false;             // Background object
input bool            InpSelection=true;         // Highlight to move
input bool            InpRayRight=false;         // Object's continuation
input bool            InpHidden=true;            // Hidden in the object l
input long            InpZOrder=0;               // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Fibonacci Extension by the given coordinates              |
//+------------------------------------------------------------------+
bool FiboExpansionCreate(const long            chart_ID=0,          // ch
                         const string          name="FiboExpansion", // ch
                         const int             sub_window=0,        // su
                         datetime              time1=0,             // fi
                         double                price1=0,            // fi
                         datetime              time2=0,             // se
                         double                price2=0,            // se
                         datetime              time3=0,             // th
                         double                price3=0,            // th
                         const color           clr=clrRed,          // ob
                         const ENUM_LINE_STYLE style=STYLE_SOLID,   // st
                         const int             width=1,             // wi
                         const bool            back=false,          // in
                         const bool            selection=true,      // hi
                         const bool            ray_right=false,     // ob
                         const bool            hidden=true,         // hi
                         const long            z_order=0)           // pr
  {
//--- set anchor points' coordinates if they are not set
   ChangeFiboExpansionEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- Create Fibonacci Extension by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_EXPANSION,sub_window,time1,price1,ti
     {
      Print(__FUNCTION__,
            ": failed to create \"Fibonacci Extension\"! Error code = ",Ge
      return(false);
     }
//--- set the object's color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
```

```
//--- set the line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the channe
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- enable (true) or disable (false) the mode of continuation of the obj
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set number of levels and their parameters                        |
//+------------------------------------------------------------------+
bool FiboExpansionLevelsSet(int             levels,               // numbe
                            double          &values[],            // value
                            color           &colors[],            // color
                            ENUM_LINE_STYLE &styles[],            // style
                            int             &widths[],            // width
                            const long      chart_ID=0,           // chart
                            const string    name="FiboExpansion") // objec
  {
//--- check array sizes
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": array length does not correspond to the number
      return(false);
     }
//--- set the number of levels
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- set the properties of levels in the loop
   for(int i=0;i<levels;i++)
     {
      //--- level value
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- level color
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- level style
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- level width
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- level description
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,"FE "+DoubleToStri
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Fibonacci Expansion anchor point                            |
//+------------------------------------------------------------------+
bool FiboExpansionPointChange(const long   chart_ID=0,           // chart'
                              const string name="FiboExpansion", // object
                              const int    point_index=0,        // anchor
                              datetime     time=0,               // anchor
                              double       price=0)              // anchor
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Fibonacci Expansion                                       |
//+------------------------------------------------------------------+
bool FiboExpansionDelete(const long   chart_ID=0,           // chart's ID
                         const string name="FiboExpansion") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
```

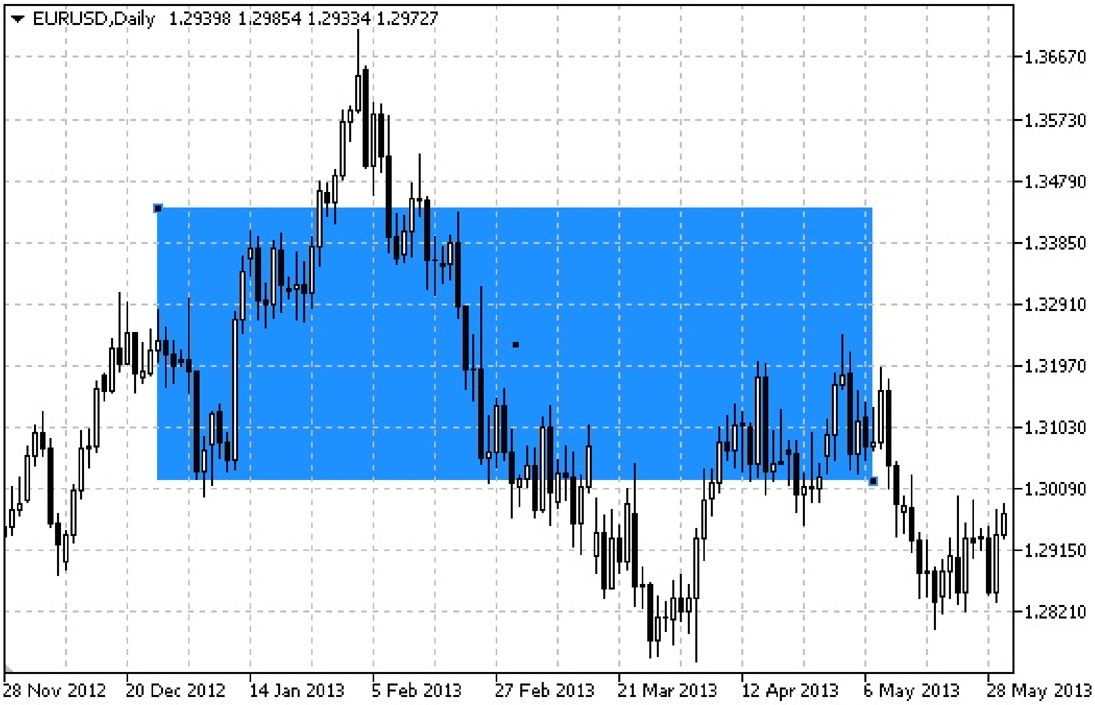
```
     {
      Print(__FUNCTION__,
            ": failed to delete \"Fibonacci Expansion\"! Error code = ",Ge
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of Fibonacci Expansion anchor points and set    |
//| default values for empty ones                                    |
//+------------------------------------------------------------------+
void ChangeFiboExpansionEmptyPoints(datetime &time1,double &price1,datetim
                                    double &price2,datetime &time3,double
  {
//--- if the third (right) point's time is not set, it will be on the curr
   if(!time3)
      time3=TimeCurrent();
//--- if the third point's price is not set, it will have Bid value
   if(!price3)
      price3=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the first (left) point's time is not set, it is located 9 bars le
//--- array for receiving the open time of the last 10 bars
   datetime temp[];
   ArrayResize(temp,10);
   if(!time1)
     {
      CopyTime(Symbol(),Period(),time3,10,temp);
      //--- set the first point 9 bars left from the second one
      time1=temp[0];
     }
//--- if the first point's price is not set, it is equal to the third poin
   if(!price1)
      price1=price3;
//--- if the second point's time is not set, it is located 7 bars left fro
   if(!time2)
      time2=temp[2];
//--- if the second point's price is not set, move it 250 points lower tha
   if(!price2)
      price2=price1-250*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
```

```
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing object anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing Fibonacci Expansion
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create Fibonacci Expansion
   if(!FiboExpansionCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayRight,InpHidde
     {
      return;
     }
//--- redraw the chart and wait for 1 second
```

```
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor points
//--- loop counter
   int v_steps=accuracy/10;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!FiboExpansionPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy/2;
//--- move the third anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p3>1)
         p3-=1;
      //--- move the point
      if(!FiboExpansionPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy*4/5;
//--- move the second anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p2<accuracy-1)
         p2+=1;
      //--- move the point
      if(!FiboExpansionPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the object from the chart
   FiboExpansionDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_RECTANGLE

Rectangle.



Example

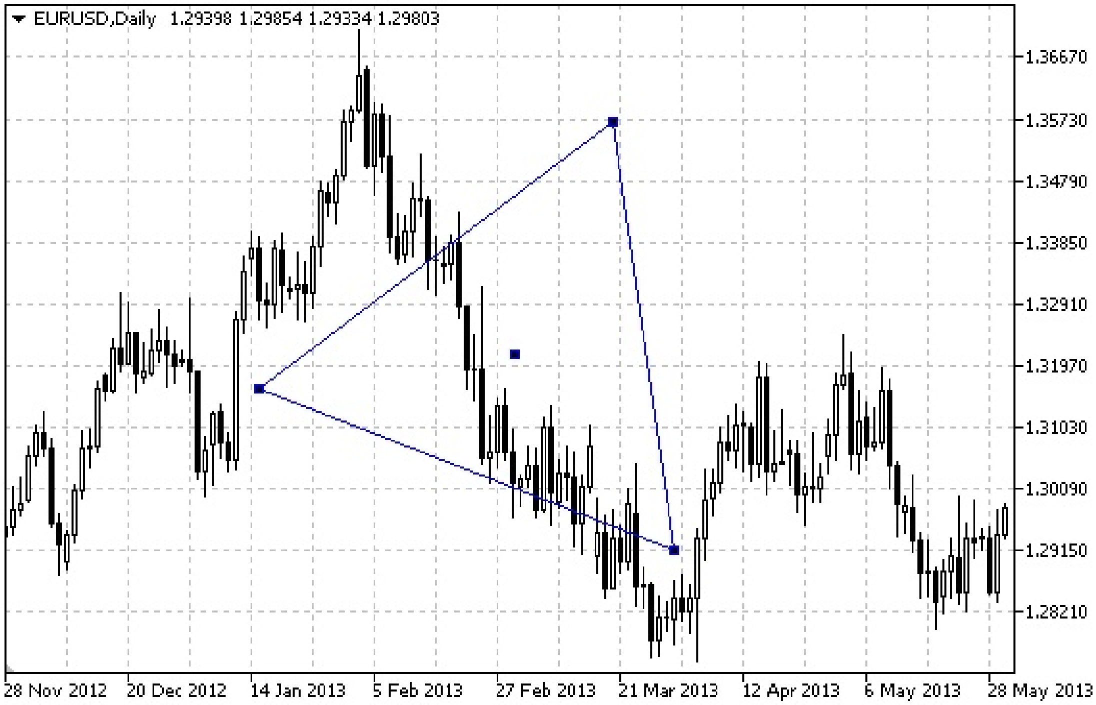
The following script creates and moves the rectangle on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates rectangle on the chart."
#property description "Anchor point coordinates are set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Rectangle"; // Rectangle name
input int             InpDate1=40;         // 1 st point's date, %
input int             InpPrice1=40;        // 1 st point's price, %
input int             InpDate2=60;         // 2 nd point's date, %
input int             InpPrice2=60;        // 2 nd point's price, %
input color           InpColor=clrRed;     // Rectangle color
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Style of rectangle lines
input int             InpWidth=1;          // Width of rectangle lines
input bool            InpFill=true;        // Filling the rectangle with c
input bool            InpBack=false;       // Background rectangle
input bool            InpSelection=true;   // Highlight to move
input bool            InpHidden=true;      // Hidden in the object list
input long            InpZOrder=0;         // Priority for mouse click
//+------------------------------------------------------------------+
//| Create rectangle by the given coordinates                        |
//+------------------------------------------------------------------+
bool RectangleCreate(const long            chart_ID=0,        // chart's I
                     const string          name="Rectangle",  // rectangle
                     const int             sub_window=0,      // subwindow
                     datetime              time1=0,           // first poi
                     double                price1=0,          // first poi
                     datetime              time2=0,           // second po
                     double                price2=0,          // second po
                     const color           clr=clrRed,        // rectangle
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // style of
                     const int             width=1,           // width of
                     const bool            fill=false,        // filling r
                     const bool            back=false,        // in the ba
                     const bool            selection=true,    // highlight
                     const bool            hidden=true,       // hidden in
                     const long            z_order=0)         // priority
  {
//--- set anchor points' coordinates if they are not set
   ChangeRectangleEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create a rectangle by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE,sub_window,time1,price1,ti
     {
      Print(__FUNCTION__,
            ": failed to create a rectangle! Error code = ",GetLastError()
      return(false);
     }
//--- set rectangle color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the style of rectangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of the rectangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the rectan
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the rectangle anchor point                                  |
//+------------------------------------------------------------------+
bool RectanglePointChange(const long   chart_ID=0,       // chart's ID
                          const string name="Rectangle", // rectangle name
                          const int    point_index=0,    // anchor point i
                          datetime     time=0,           // anchor point t
                          double       price=0)          // anchor point p
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the rectangle                                             |
//+------------------------------------------------------------------+
bool RectangleDelete(const long   chart_ID=0,       // chart's ID
                     const string name="Rectangle") // rectangle name
  {
//--- reset the error value
   ResetLastError();
//--- delete rectangle
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete rectangle! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of rectangle's anchor points and set default    |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeRectangleEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left fro
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, move it 300 points lower tha
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
```

```
//--- arrays for storing the date and price values to be used
//--- for setting and changing rectangle anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the rectangle
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create a rectangle
   if(!RectangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],I
      InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the rectangle's anchor points
//--- loop counter
   int h_steps=bars/2;
//--- move the anchor points
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following values
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- shift the points
      if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int v_steps=accuracy/2;
//--- move the anchor points
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following values
      if(p1<accuracy-1)
         p1+=1;
      if(p2>1)
         p2-=1;
      //--- shift the points
      if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the rectangle from the chart
   RectangleDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_TRIANGLE

Triangle.



Example

The following script creates and moves the triangle on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates triangle on the chart."
#property description "Anchor point coordinates are set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Triangle";         // Triangle name
input int             InpDate1=25;                // 1 st point's date, %
input int             InpPrice1=50;               // 1 st point's price, %
input int             InpDate2=70;                // 2 nd point's date, %
input int             InpPrice2=70;               // 2 nd point's price, %
input int             InpDate3=65;                // 3 rd point's date, %
input int             InpPrice3=20;               // 3 rd point's price, %
input color           InpColor=clrRed;            // Triangle color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of triangle line
input int             InpWidth=1;                // Width of triangle line
input bool            InpFill=false;             // Filling triangle with
input bool            InpBack=false;             // Background triangle
input bool            InpSelection=true;         // Highlight to move
input bool            InpHidden=true;            // Hidden in the object l
input long            InpZOrder=0;               // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create triangle by the given coordinates                         |
//+------------------------------------------------------------------+
bool TriangleCreate(const long            chart_ID=0,        // chart's ID
                    const string          name="Triangle",   // triangle r
                    const int             sub_window=0,      // subwindow
                    datetime              time1=0,           // first poin
                    double                price1=0,          // first poin
                    datetime              time2=0,           // second poi
                    double                price2=0,          // second poi
                    datetime              time3=0,           // third poin
                    double                price3=0,          // third poin
                    const color           clr=clrRed,        // triangle c
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // style of t
                    const int             width=1,           // width of t
                    const bool            fill=false,        // filling tr
                    const bool            back=false,        // in the bac
                    const bool            selection=true,    // highlight
                    const bool            hidden=true,       // hidden in
                    const long            z_order=0)         // priority f
  {
//--- set anchor points' coordinates if they are not set
   ChangeTriangleEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- reset the error value
   ResetLastError();
//--- create triangle by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_TRIANGLE,sub_window,time1,price1,tim
     {
      Print(__FUNCTION__,
            ": failed to create a triangle! Error code = ",GetLastError())
      return(false);
     }
//--- set triangle color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of triangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of triangle lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the triang
```

```
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the triangle anchor point                                   |
//+------------------------------------------------------------------+
bool TrianglePointChange(const long   chart_ID=0,       // chart's ID
                         const string name="Triangle", // triangle name
                         const int    point_index=0,   // anchor point ind
                         datetime     time=0,          // anchor point tim
                         double       price=0)         // anchor point pri
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the triangle                                              |
//+------------------------------------------------------------------+
bool TriangleDelete(const long   chart_ID=0,       // chart's ID
                    const string name="Triangle") // triangle name
  {
//--- reset the error value
   ResetLastError();
//--- delete the triangle
```

```
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the ellipse! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of triangle's anchor points and set default     |
//| values for empty ones                                            |
//+------------------------------------------------------------------+
void ChangeTriangleEmptyPoints(datetime &time1,double &price1,
                               datetime &time2,double &price2,
                               datetime &time3,double &price3)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left fro
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, move it 300 points lower tha
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- if the third point's time is not set, it coincides with the second p
   if(!time3)
      time3=time2;
//--- if the third point's price is not set, it is equal to the first poin
   if(!price3)
      price3=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
```

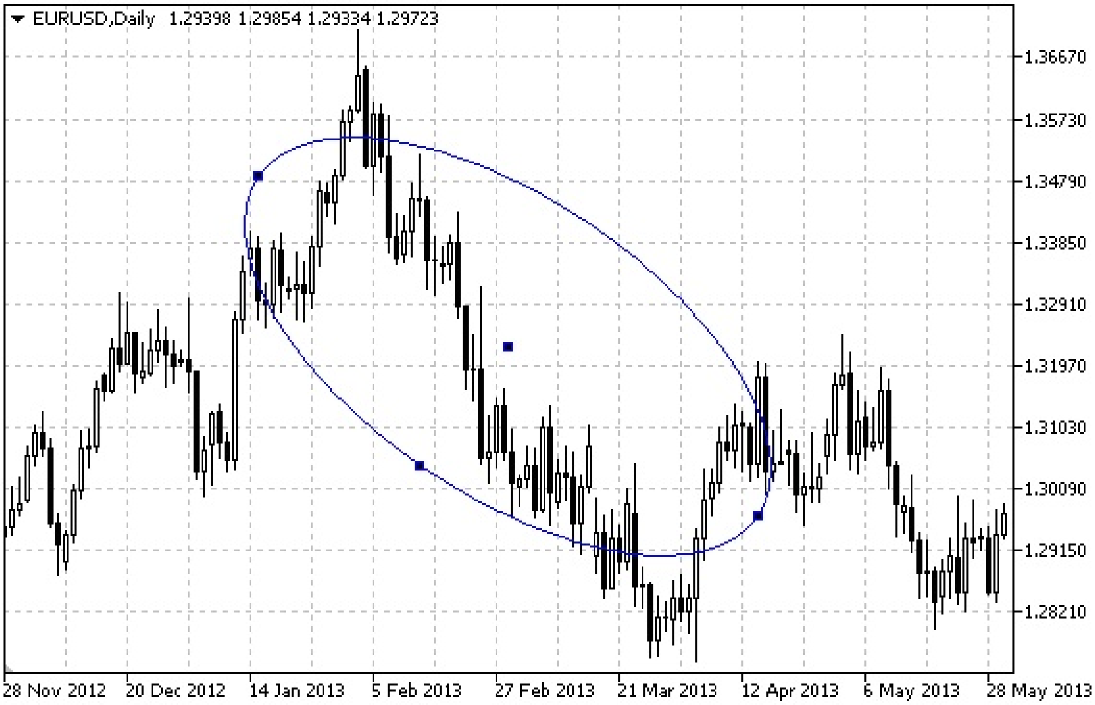
```
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100 ||
      InpDate3<0 || InpDate3>100 || InpPrice3<0 || InpPrice3>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing triangle anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the triangle
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
//--- create a triangle
   if(!TriangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],da
      InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,In
     {
      return;
     }
```

```
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the ellipse anchor points
//--- loop counter
   int v_steps=accuracy*3/10;
//--- move the first anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p1>1)
         p1-=1;
      //--- move the point
      if(!TrianglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int h_steps=bars*9/20-1;
//--- move the second anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d2>1)
         d2-=1;
      //--- move the point
      if(!TrianglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   v_steps=accuracy/4;
//--- move the third anchor point
```

```
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p3<accuracy-1)
         p3+=1;
      //--- move the point
      if(!TrianglePointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete triangle from the chart
   TriangleDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ELLIPSE

Ellipse.



Example

The following script creates and moves the ellipse on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

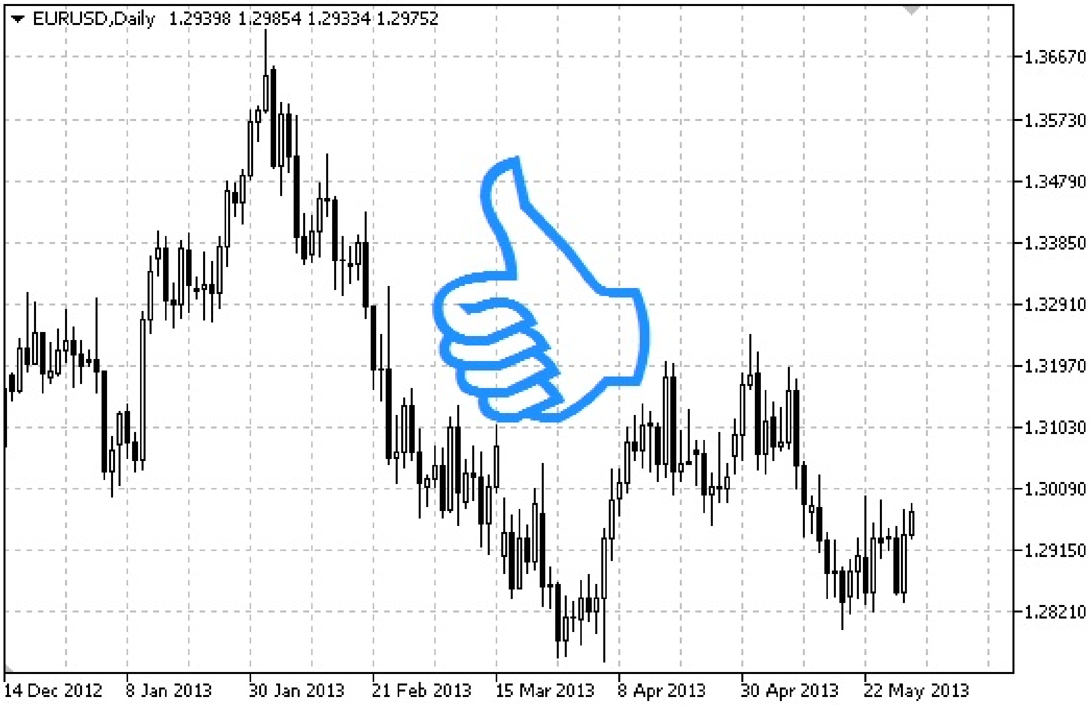
```
#property strict //--- description
#property description "Script creates ellipse on the chart."
#property description "Anchor point coordinates are set"
#property description "in percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Ellipse";          // Ellipse name
input int             InpDate1=30;                // 1 st point's date, %
input int             InpPrice1=20;               // 1 st point's price, %
input int             InpDate2=70;                // 2 nd point's date, %
input int             InpPrice2=80;               // 2 nd point's price, %
input double          InpEllipseScale=0.2;        // Ellipse scale ratio
input color           InpColor=clrRed;            // Ellipse color
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Style of ellipse lines
input int             InpWidth=1;                 // Width of ellipse lines
input bool            InpFill=false;              // Filling ellipse with c
input bool            InpBack=false;              // Background ellipse
input bool            InpSelection=true;          // Highlight to move
input bool            InpHidden=true;             // Hidden in the object l
input long            InpZOrder=0;                // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create an ellipse by the given coordinates                       |
//+------------------------------------------------------------------+
bool EllipseCreate(const long            chart_ID=0,        // chart's ID
                   const string          name="Ellipse",    // ellipse nam
                   const int             sub_window=0,      // subwindow i
                   datetime              time1=0,           // first point
                   double                price1=0,          // first point
                   datetime              time2=0,           // second poi
                   double                price2=0,          // second poi
                   double                ellipse_scale=0,   // ellipse sca
                   const color           clr=clrRed,        // ellipse col
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // style of el
                   const int             width=1,           // width of el
                   const bool            fill=false,        // filling ell
                   const bool            back=false,        // in the back
                   const bool            selection=true,    // highlight t
                   const bool            hidden=true,       // hidden in t
                   const long            z_order=0)         // priority fo
  {
//--- set anchor points' coordinates if they are not set
   ChangeEllipseEmptyPoints(time1,price1,time2,price2);
//--- reset the error value
   ResetLastError();
//--- create an ellipse by the given coordinates
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIPSE,sub_window,time1,price1,time
     {
      Print(__FUNCTION__,
            ": failed to create an ellipse! Error code = ",GetLastError())
      return(false);
     }
//--- set ellipse scale ratio
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,InpEllipseScale);
//--- set an ellipse color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set style of ellipse lines
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set width of ellipse lines
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of highlighting the ellips
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the ellipse anchor point                                    |
//+------------------------------------------------------------------+
bool EllipsePointChange(const long   chart_ID=0,     // chart's ID
                        const string name="Ellipse", // ellipse name
                        const int    point_index=0,  // anchor point index
                        datetime     time=0,         // anchor point time
                        double       price=0)        // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete ellipse                                                   |
//+------------------------------------------------------------------+
bool EllipseDelete(const long   chart_ID=0,     // chart's ID
                   const string name="Ellipse") // ellipse name
  {
//--- reset the error value
   ResetLastError();
//--- delete an ellipse
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete an ellipse! Error code = ",GetLastError())
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check the values of ellipse anchor points and set default values |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeEllipseEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
  {
//--- if the first point's time is not set, it will be on the current bar
   if(!time1)
      time1=TimeCurrent();
//--- if the first point's price is not set, it will have Bid value
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- if the second point's time is not set, it is located 9 bars left fro
   if(!time2)
     {
      //--- array for receiving the open time of the last 10 bars
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- set the second point 9 bars left from the first one
      time2=temp[0];
     }
//--- if the second point's price is not set, move it 300 points lower tha
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
```

```
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing ellipse anchor points' coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- set as series
   ArraySetAsSeries(date,true);
   ArraySetAsSeries(price,true);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the ellipse
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- create an ellipse
   if(!EllipseCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],Inp
      InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,In
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the ellipse anchor points
//--- loop counter
   int v_steps=accuracy/5;
//--- move the first and second anchor points
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following values
      if(p1<accuracy-1)
         p1+=1;
      if(p2>1)
         p2-=1;
      //--- shift the points
      if(!EllipsePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!EllipsePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      //--- delay
      Sleep(10);
     }
//--- 1 second of delay
   Sleep(1000);
//--- inital ellipse scale ratio
   double ellipse_scale=InpEllipseScale;
//--- loop counter
   v_steps=100;
//--- change ellipse scale ratio
   for(int i=0;i<v_steps;i++)
     {
      ellipse_scale+=2.0/v_steps;
      //--- set scale ratio
      if(!ObjectSetDouble(0,InpName,OBJPROP_SCALE,ellipse_scale)) return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      //--- delay
      Sleep(20);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete ellipse from the chart
   EllipseDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_THUMB_UP

Thumbs Up sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Thumbs Up sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Thumbs Up\" sign."
#property description "Anchor point coordinate is set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ThumbUp";     // Sign name
input int               InpDate=75;            // Anchor point date in %
input int               InpPrice=25;          // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Anchor type
input color             InpColor=clrRed;      // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;   // Border line style
input int               InpWidth=5;           // Sign size
input bool              InpBack=false;        // Background sign
input bool              InpSelection=true;    // Highlight to move
input bool              InpHidden=true;       // Hidden in the object list
input long              InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Thumbs Up sign                                            |
//+------------------------------------------------------------------+
bool ArrowThumbUpCreate(const long              chart_ID=0,           // c
                        const string            name="ThumbUp",       // s
                        const int               sub_window=0,         // s
                        datetime                time=0,               // a
                        double                  price=0,              // a
                        const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // a
                        const color             clr=clrRed,           // s
                        const ENUM_LINE_STYLE   style=STYLE_SOLID,    // b
                        const int               width=3,              // s
                        const bool              back=false,           // i
                        const bool              selection=true,       // h
                        const bool              hidden=true,          // h
                        const long              z_order=0)            // p
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_UP,sub_window,time,price
     {
      Print(__FUNCTION__,
            ": failed to create \"Thumbs Up\" sign! Error code = ",GetLast
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

```
//--- enable (true) or disable (false) the mode of moving the sign by mous
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowThumbUpMove(const long   chart_ID=0,     // chart's ID
                      const string name="ThumbUp", // object name
                      datetime     time=0,         // anchor point time co
                      double       price=0)        // anchor point price c
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Thumbs Up sign anchor type                                |
//+------------------------------------------------------------------+
bool ArrowThumbUpAnchorChange(const long              chart_ID=0,        /
                              const string            name="ThumbUp",    /
                              const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) /
  {
//--- reset the error value
   ResetLastError();
```

```
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Thumbs Up sign                                            |
//+------------------------------------------------------------------+
bool ArrowThumbUpDelete(const long   chart_ID=0,      // chart's ID
                        const string name="ThumbUp") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Thumbs Up\" sign! Error code = ",GetLast
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
```

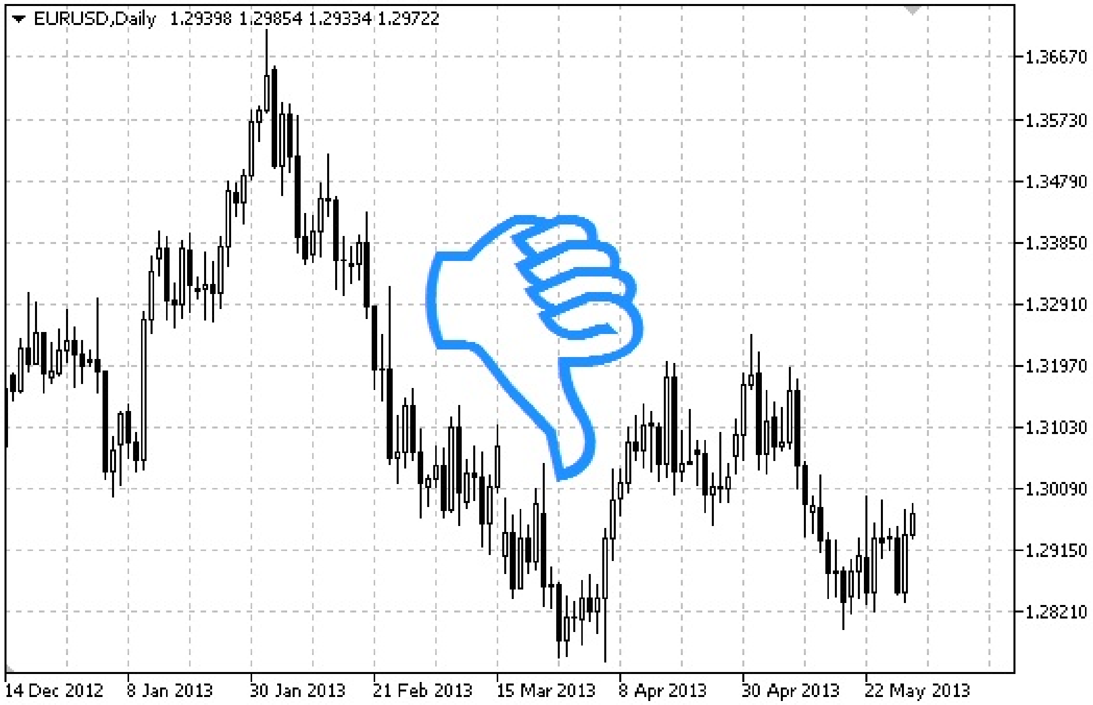
```
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Thumbs Up sign on the chart
   if(!ArrowThumbUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the s
//--- loop counter
   int h_steps=bars/4;
//--- move the anchor point
```

```
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d>1)
         d-=1;
      //--- move the point
      if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int v_steps=accuracy/4;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p<accuracy-1)
         p+=1;
      //--- move the point
      if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- change anchor point location relative to the sign
   ArrowThumbUpAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowThumbUpDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
```

```
    }
```

OBJ_ARROW_THUMB_DOWN

Thumbs Down sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Thumbs Down sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Thumbs Down\" sign."
#property description "Anchor point coordinate is set in percentage of"
#property description "the chart window size."
//--- display window of the input parameters during the script's launch
```

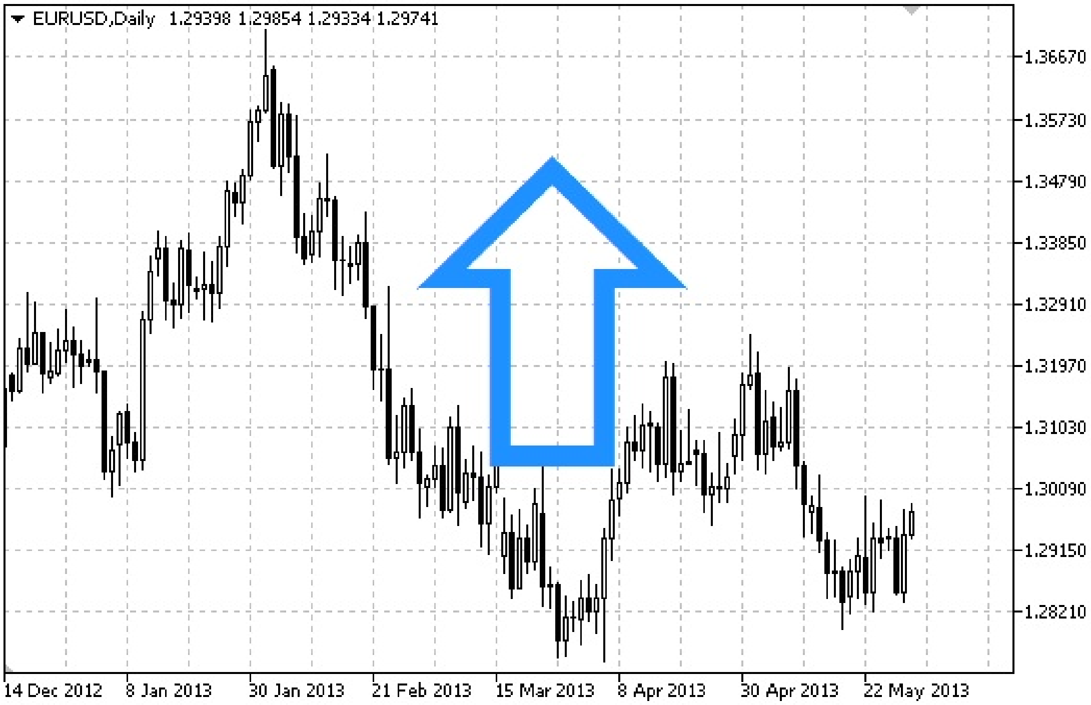
```
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ThumbDown";     // Sign name
input int               InpDate=25;              // Anchor point date in %
input int               InpPrice=75;             // Anchor point price in
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Anchor type
input color             InpColor=clrRed;         // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;      // Border line style
input int               InpWidth=5;              // Sign size
input bool              InpBack=false;           // Background sign
input bool              InpSelection=true;       // Highlight to move
input bool              InpHidden=true;          // Hidden in the object l
input long              InpZOrder=0;             // Priority for mouse cli
//+-------------------------------------------------------------------+
//| Create Thumbs Down sign                                           |
//+-------------------------------------------------------------------+
bool ArrowThumbDownCreate(const long              chart_ID=0,           //
                          const string            name="ThumbDown",     //
                          const int               sub_window=0,         //
                          datetime                time=0,               //
                          double                  price=0,              //
                          const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, //
                          const color             clr=clrRed,           //
                          const ENUM_LINE_STYLE   style=STYLE_SOLID,    //
                          const int               width=3,              //
                          const bool              back=false,           //
                          const bool              selection=true,       //
                          const bool              hidden=true,          //
                          const long              z_order=0)            //
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_DOWN,sub_window,time,pri
     {
      Print(__FUNCTION__,
            ": failed to create \"Thumbs Down\" sign! Error code = ",GetLa
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
```

```
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mous
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowThumbDownMove(const long   chart_ID=0,       // chart's ID
                        const string name="ThumbDown", // object name
                        datetime     time=0,           // anchor point tim
                        double       price=0)          // anchor point pri
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Thumbs Down sign anchor type                              |
//+------------------------------------------------------------------+
bool ArrowThumbDownAnchorChange(const long                chart_ID=0,
                                const string              name="ThumbDown",
                                const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP)
  {
//--- reset the error value
   ResetLastError();
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Thumbs Down sign                                          |
//+------------------------------------------------------------------+
bool ArrowThumbDownDelete(const long   chart_ID=0,       // chart's ID
                          const string name="ThumbDown") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Thumbs Down\" sign! Error code = ",GetLa
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Thumbs Down sign on the chart
   if(!ArrowThumbDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColc
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
```

```
//--- now, move the anchor point and change its position relative to the s
//--- loop counter
   int h_steps=bars/4;
//--- move the anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- loop counter
   int v_steps=accuracy/4;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p>1)
         p-=1;
      //--- move the point
      if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- change anchor point location relative to the sign
   ArrowThumbDownAnchorChange(0,InpName,ANCHOR_TOP);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowThumbDownDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_UP

Arrow Up sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Arrow Up sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Arrow Up\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ArrowUp";     // Sign name
input int               InpDate=25;            // Anchor point date in %
input int               InpPrice=25;          // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Anchor type
input color             InpColor=clrRed;      // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;   // Border line style
input int               InpWidth=5;           // Sign size
input bool              InpBack=false;        // Background sign
input bool              InpSelection=false;   // Highlight to move
input bool              InpHidden=true;       // Hidden in the object list
input long              InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Array Up sign                                             |
//+------------------------------------------------------------------+
bool ArrowUpCreate(const long              chart_ID=0,           // chart'
                   const string            name="ArrowUp",       // sign n
                   const int               sub_window=0,         // subwin
                   datetime                time=0,               // anchor
                   double                  price=0,              // anchor
                   const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor
                   const color             clr=clrRed,           // sign c
                   const ENUM_LINE_STYLE   style=STYLE_SOLID,    // border
                   const int               width=3,              // sign s
                   const bool              back=false,           // in the
                   const bool              selection=true,       // highli
                   const bool              hidden=true,          // hidden
                   const long              z_order=0)            // priori
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_UP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Arrow Up\" sign! Error code = ",GetLastE
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

```
//--- enable (true) or disable (false) the mode of moving the sign by mous
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowUpMove(const long   chart_ID=0,     // chart's ID
                 const string name="ArrowUp", // object name
                 datetime     time=0,         // anchor point time coordin
                 double       price=0)        // anchor point price coordi
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Arrow Down sign anchor type                               |
//+------------------------------------------------------------------+
bool ArrowUpAnchorChange(const long              chart_ID=0,        // cha
                         const string            name="ArrowUp",    // obj
                         const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // anc
  {
//--- reset the error value
   ResetLastError();
```

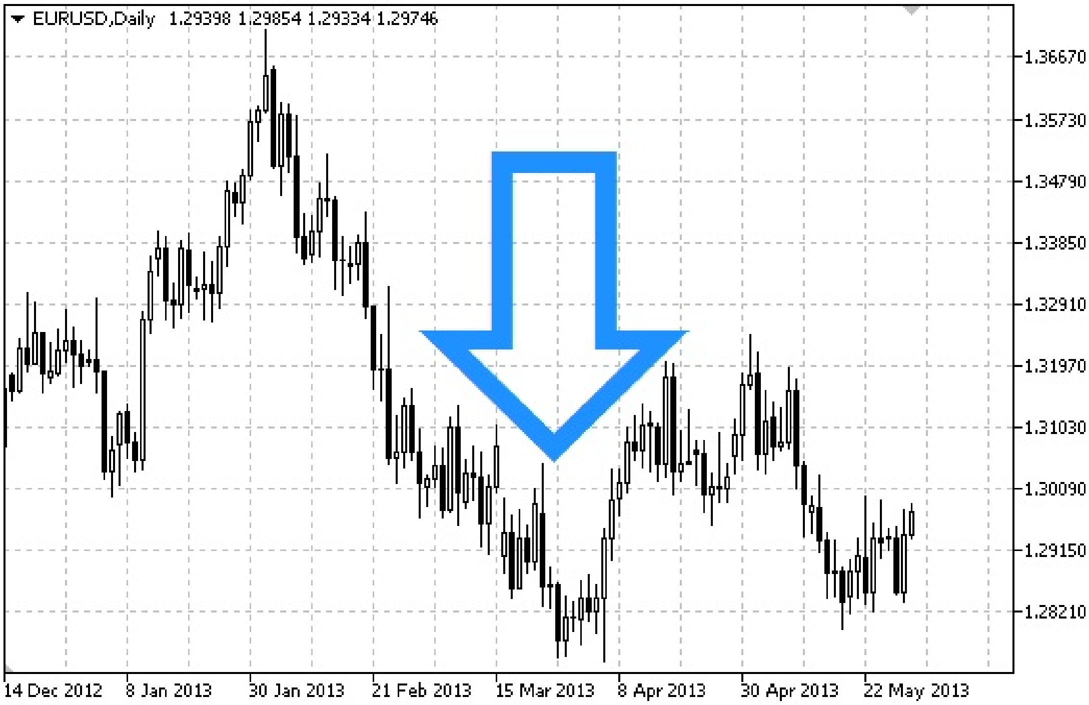
```
//--- change anchor point location
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Arrow Up sign                                             |
//+------------------------------------------------------------------+
bool ArrowUpDelete(const long   chart_ID=0,     // chart's ID
                   const string name="ArrowUp") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Arrow Up\" sign! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
```

```
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Arrow Up sign on the chart
   if(!ArrowUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the s
//--- loop counter
   int v_steps=accuracy/2;
//--- move the anchor point
```

```
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p<accuracy-1)
         p+=1;
      //--- move the point
      if(!ArrowUpMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- change anchor point location relative to the sign
   ArrowUpAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowUpDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_DOWN

Arrow Down sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Arrow Down sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Array Down\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ArrowDown";     // Sign name
input int               InpDate=75;              // Anchor point date in %
input int               InpPrice=75;             // Anchor point price in
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Anchor type
input color             InpColor=clrRed;         // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;      // Border line style
input int               InpWidth=5;              // Sign size
input bool              InpBack=false;           // Background sign
input bool              InpSelection=false;      // Highlight to move
input bool              InpHidden=true;          // Hidden in the object l
input long              InpZOrder=0;             // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Array Down sign                                           |
//+------------------------------------------------------------------+
bool ArrowDownCreate(const long              chart_ID=0,           // char
                     const string            name="ArrowDown",     // sigr
                     const int               sub_window=0,         // subw
                     datetime                time=0,               // anch
                     double                  price=0,              // anch
                     const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anch
                     const color             clr=clrRed,           // sigr
                     const ENUM_LINE_STYLE   style=STYLE_SOLID,    // bord
                     const int               width=3,              // sigr
                     const bool              back=false,           // in t
                     const bool              selection=true,       // high
                     const bool              hidden=true,          // hidd
                     const long              z_order=0)            // pric
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_DOWN,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Arrow Down\" sign! Error code = ",GetLas
      return(false);
     }
//--- anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mous
```

```
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowDownMove(const long   chart_ID=0,        // chart's ID
                   const string name="ArrowDown", // object name
                   datetime     time=0,           // anchor point time coo
                   double       price=0)          // anchor point price co
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Arrow Down sign anchor type                                |
//+------------------------------------------------------------------+
bool ArrowDownAnchorChange(const long              chart_ID=0,        // c
                           const string            name="ArrowDown",  // c
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // a
  {
//--- reset the error value
   ResetLastError();
//--- change anchor point location
```

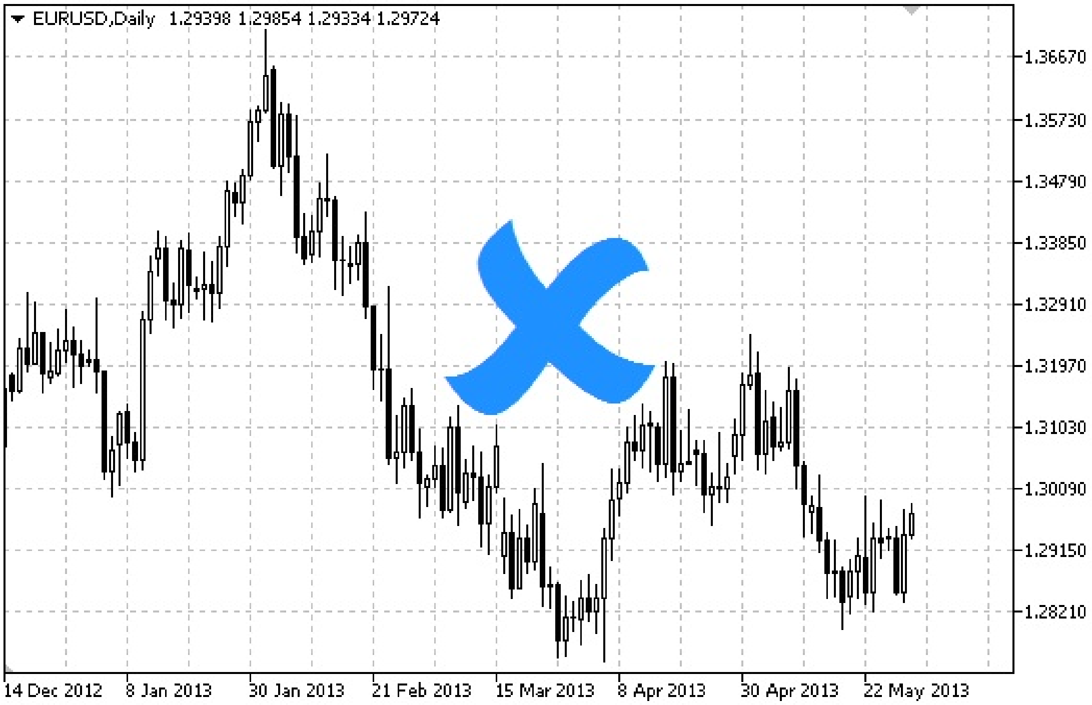
```
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Array Down sign                                           |
//+------------------------------------------------------------------+
bool ArrowDownDelete(const long   chart_ID=0,       // chart's ID
                     const string name="ArrowDown") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Arrow Down\" sign! Error code = ",GetLas
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
```

```
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Arrow Down sign on the chart
   if(!ArrowDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the s
//--- loop counter
   int v_steps=accuracy/2;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
```

```
     {
      //--- use the following value
      if(p>1)
         p-=1;
      //--- move the point
      if(!ArrowDownMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- change anchor point location relative to the sign
   ArrowDownAnchorChange(0,InpName,ANCHOR_TOP);
//--- redraw the chart
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowDownDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_STOP

Stop sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Stop sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Stop\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ArrowStop";     // Sign name
input int               InpDate=10;              // Anchor point date in
input int               InpPrice=50;            // Anchor point price in
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Anchor type
input color             InpColor=clrRed;        // Sign color
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;     // Border line style
input int               InpWidth=5;             // Sign size
input bool              InpBack=false;          // Background sign
input bool              InpSelection=false;     // Highlight to move
input bool              InpHidden=true;         // Hidden in the object l
input long              InpZOrder=0;            // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create Stop sign                                                 |
//+------------------------------------------------------------------+
bool ArrowStopCreate(const long              chart_ID=0,           // char
                     const string            name="ArrowStop",     // sign
                     const int               sub_window=0,         // subw
                     datetime                time=0,               // anch
                     double                  price=0,              // anch
                     const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anch
                     const color             clr=clrRed,           // sign
                     const ENUM_LINE_STYLE   style=STYLE_SOLID,    // bord
                     const int               width=3,              // sign
                     const bool              back=false,           // in t
                     const bool              selection=true,       // high
                     const bool              hidden=true,          // hidd
                     const long              z_order=0)            // pric
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_STOP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Stop\" sign! Error code = ",GetLastError
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

```
//--- enable (true) or disable (false) the mode of moving the sign by mous
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowStopMove(const long   chart_ID=0,        // chart's ID
                   const string name="ArrowStop", // object name
                   datetime     time=0,            // anchor point time coo
                   double       price=0)           // anchor point price co
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Stop sign anchor type                                     |
//+------------------------------------------------------------------+
bool ArrowStopAnchorChange(const long              chart_ID=0,         // c
                           const string            name="ArrowStop",   // o
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // a
  {
//--- reset the error value
   ResetLastError();
```

```
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Stop sign                                                 |
//+------------------------------------------------------------------+
bool ArrowStopDelete(const long   chart_ID=0,       // chart's ID
                     const string name="ArrowStop") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Stop\" sign! Error code = ",GetLastError
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
```

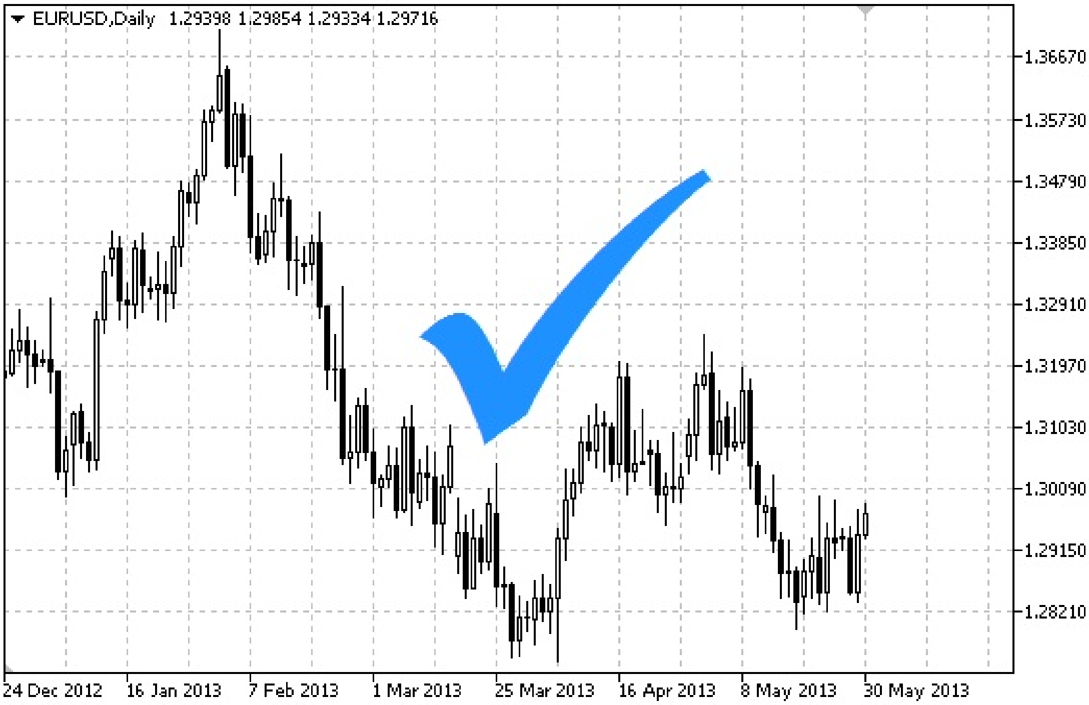
```
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Stop sign on the chart
   if(!ArrowStopCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the s
//--- loop counter
   int h_steps=bars*2/5;
//--- move the anchor point
```

```
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!ArrowStopMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.025 seconds of delay
      Sleep(25);
     }
//--- change anchor point location relative to the sign
   ArrowStopAnchorChange(0,InpName,ANCHOR_TOP);
//--- redraw the chart
   ChartRedraw();
//--- loop counter
   h_steps=bars*2/5;
//--- move the anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!ArrowStopMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.025 seconds of delay
      Sleep(25);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowStopDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
```

```
    }
```

OBJ_ARROW_CHECK

Check sign.



Note

Anchor point position relative to the sign can be selected from ENUM_ARROW_ANCHOR enumeration.

Large signs (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

Example

The following script creates and moves Check sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Check\" sign."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="ArrowCheck"; // Sign name
input int               InpDate=10;           // Anchor point date in %
input int                 InpPrice=50;          // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Anchor type
input color               InpColor=clrRed;      // Sign color
input ENUM_LINE_STYLE    InpStyle=STYLE_DOT;   // Border line style
input int                 InpWidth=5;           // Sign size
input bool                InpBack=false;        // Background sign
input bool                InpSelection=false;   // Highlight to move
input bool                InpHidden=true;       // Hidden in the object list
input long                InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Check sign                                                |
//+------------------------------------------------------------------+
bool ArrowCheckCreate(const long              chart_ID=0,           // cha
                      const string            name="ArrowCheck",    // sig
                      const int               sub_window=0,         // sub
                      datetime                time=0,               // anc
                      double                  price=0,              // anc
                      const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anc
                      const color             clr=clrRed,           // sig
                      const ENUM_LINE_STYLE   style=STYLE_SOLID,    // bor
                      const int               width=3,              // sig
                      const bool              back=false,           // in
                      const bool              selection=true,       // hig
                      const bool              hidden=true,          // hid
                      const long              z_order=0)            // pri
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_CHECK,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Check\" sign! Error code = ",GetLastErro
      return(false);
     }
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the sign size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

```
//--- enable (true) or disable (false) the mode of moving the sign by mous
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowCheckMove(const long   chart_ID=0,        // chart's ID
                    const string name="ArrowCheck", // object name
                    datetime     time=0,            // anchor point time c
                    double       price=0)           // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Check anchor type                                         |
//+------------------------------------------------------------------+
bool ArrowCheckAnchorChange(const long              chart_ID=0,        //
                            const string            name="ArrowCheck", //
                            const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) //
  {
//--- reset the error value
   ResetLastError();
```

```
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Check sign                                                |
//+------------------------------------------------------------------+
bool ArrowCheckDelete(const long   chart_ID=0,        // chart's ID
                      const string name="ArrowCheck") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Check\" sign! Error code = ",GetLastErrc
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
```

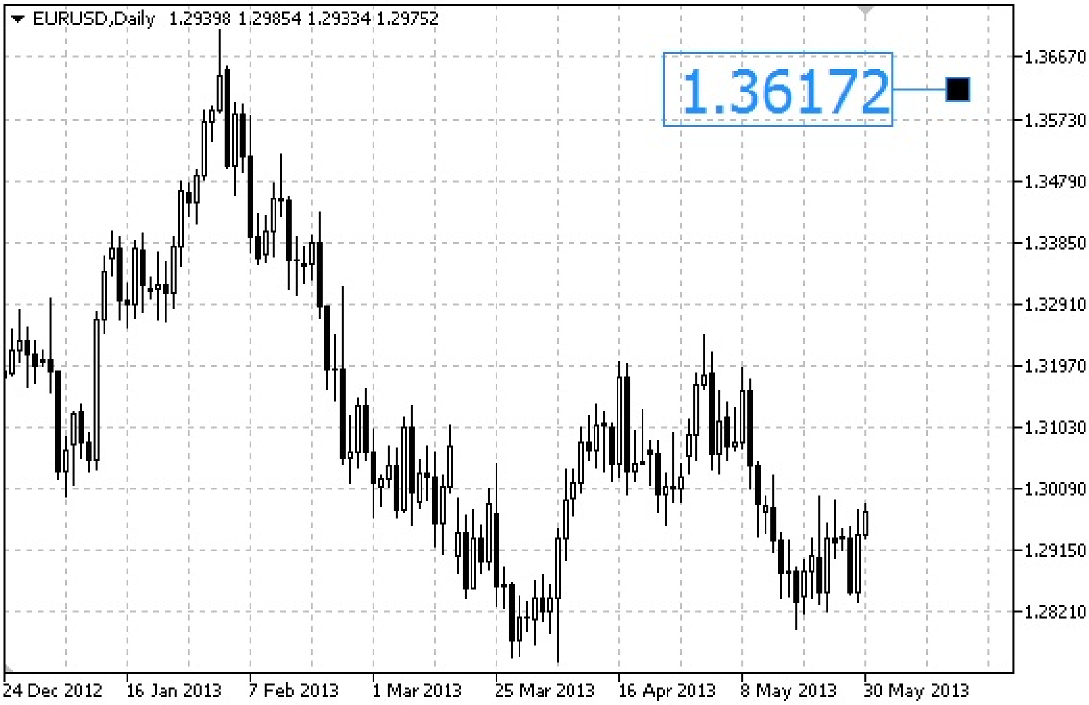
```
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the sign
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create Check sign on the chart
   if(!ArrowCheckCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point and change its position relative to the s
//--- loop counter
   int h_steps=bars*2/5;
//--- move the anchor point
```

```
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!ArrowCheckMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.025 seconds of delay
      Sleep(25);
     }
//--- change anchor point location relative to the sign
   ArrowCheckAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- redraw the chart
   ChartRedraw();
//--- loop counter
   h_steps=bars*2/5;
//--- move the anchor point
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!ArrowCheckMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.025 seconds of delay
      Sleep(25);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the sign from the chart
   ArrowCheckDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
```

```
    }
```

OBJ_ARROW_LEFT_PRICE

Left Price Label



Example

The following script creates and moves left price label on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

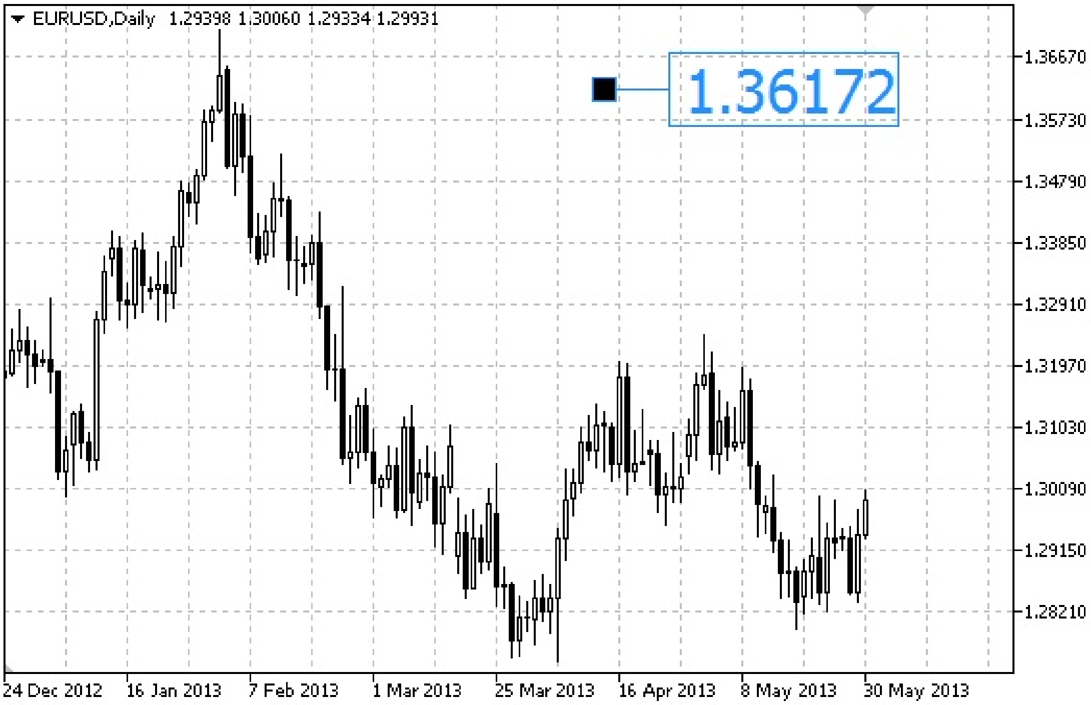
```
#property strict //--- description
#property description "Script creates the left price label on the chart."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="LeftPrice";   // Price label name
input int               InpDate=100;           // Anchor point date in %
input int               InpPrice=10;           // Anchor point price in %
input color             InpColor=clrRed;       // Price label color
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID; // Border line style
input int               InpWidth=2;            // Price label size
input bool              InpBack=false;         // Background label
input bool              InpSelection=true;     // Highlight to move
input bool              InpHidden=true;        // Hidden in the object list
input long              InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the left price label                                      |
//+------------------------------------------------------------------+
bool ArrowLeftPriceCreate(const long            chart_ID=0,        // char
                          const string          name="LeftPrice",  // pric
                          const int             sub_window=0,      // subw
                          datetime              time=0,            // anch
                          double                price=0,           // anch
                          const color           clr=clrRed,        // pric
                          const ENUM_LINE_STYLE style=STYLE_SOLID, // bord
                          const int             width=1,           // pric
                          const bool            back=false,        // in t
                          const bool            selection=true,    // high
                          const bool            hidden=true,       // hidd
                          const long            z_order=0)         // pric
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create a price label
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_LEFT_PRICE,sub_window,time,pri
     {
      Print(__FUNCTION__,
            ": failed to create the left price label! Error code = ",GetLa
      return(false);
     }
//--- set the label color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the label size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mou
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowLeftPriceMove(const long   chart_ID=0,       // chart's ID
                        const string name="LeftPrice", // label name
                        datetime     time=0,           // anchor point tim
                        double       price=0)          // anchor point pri
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the left price label from the chart                       |
//+------------------------------------------------------------------+
bool ArrowLeftPriceDelete(const long   chart_ID=0,       // chart's ID
                          const string name="LeftPrice") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the left price label! Error code = ",GetLa
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing label anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
```

```
      price[i]=min_price+i*step;
//--- define points for drawing the label
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create the left price label on the chart
   if(!ArrowLeftPriceCreate(0,InpName,0,date[d],price[p],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point
//--- loop counter
   int v_steps=accuracy*4/5;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p<accuracy-1)
         p+=1;
      //--- move the point
      if(!ArrowLeftPriceMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the label from the chart
   ArrowLeftPriceDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_RIGHT_PRICE

Right Price Label.



Example

The following script creates and moves right price label on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

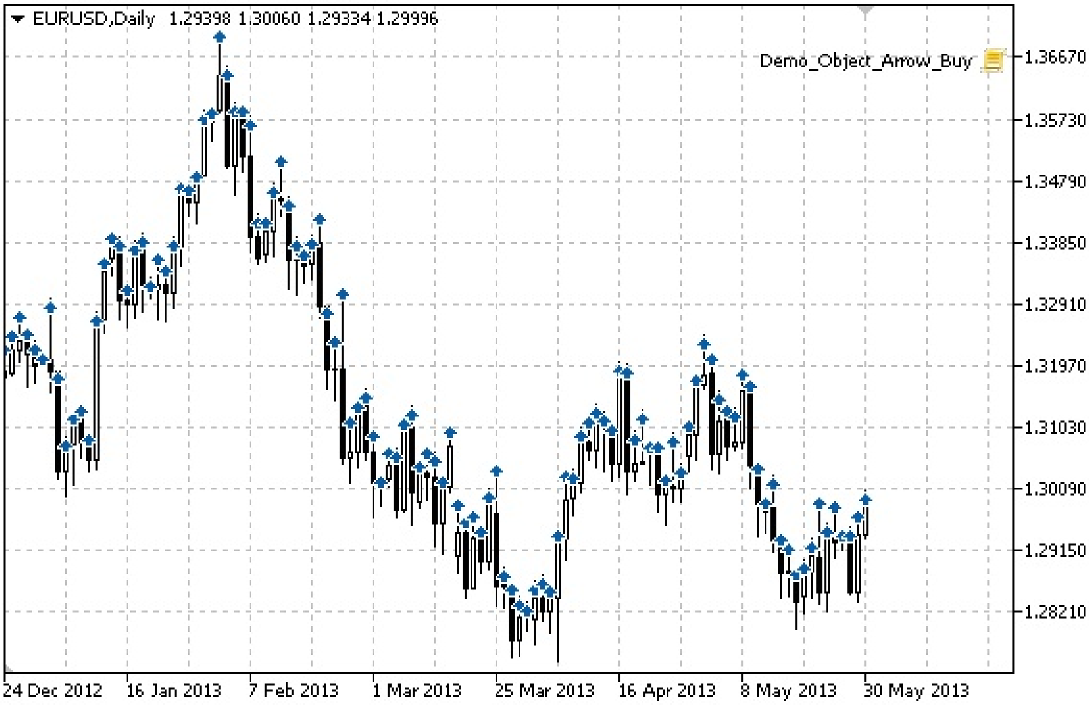
```
#property strict //--- description
#property description "Script creates the right price label on the chart."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="RightPrice"; // Price label name
input int               InpDate=0;            // Anchor point date in %
input int               InpPrice=90;          // Anchor point price in %
input color             InpColor=clrRed;      // Price label color
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID; // Border line style
input int               InpWidth=2;           // Price label size
input bool              InpBack=false;        // Background label
input bool              InpSelection=true;    // Highlight to move
input bool              InpHidden=true;       // Hidden in the object list
input long              InpZOrder=0;          // Priority for mouse click
//+------------------------------------------------------------------+
//| Create the right price label                                     |
//+------------------------------------------------------------------+
bool ArrowRightPriceCreate(const long            chart_ID=0,        // cha
                           const string          name="RightPrice", // pri
                           const int             sub_window=0,      // sub
                           datetime              time=0,            // anc
                           double                price=0,           // anc
                           const color           clr=clrRed,        // pri
                           const ENUM_LINE_STYLE style=STYLE_SOLID, // bor
                           const int             width=1,           // pri
                           const bool            back=false,        // in
                           const bool            selection=true,    // hig
                           const bool            hidden=true,       // hid
                           const long            z_order=0)         // pri
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create a price label
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_RIGHT_PRICE,sub_window,time,pr
     {
      Print(__FUNCTION__,
            ": failed to create the right price label! Error code = ",GetL
      return(false);
     }
//--- set the label color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the label size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mou
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowRightPriceMove(const long   chart_ID=0,        // chart's ID
                         const string name="RightPrice", // label name
                         datetime     time=0,            // anchor point t
                         double       price=0)           // anchor point p
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the right price label from the chart                      |
//+------------------------------------------------------------------+
bool ArrowRightPriceDelete(const long   chart_ID=0,        // chart's ID
                           const string name="RightPrice") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the right price label! Error code = ",GetL
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing label anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
```

```
      price[i]=min_price+i*step;
//--- define points for drawing the label
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create the right price label on the chart
   if(!ArrowRightPriceCreate(0,InpName,0,date[d],price[p],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the anchor point
//--- loop counter
   int v_steps=accuracy*4/5;
//--- move the anchor point
   for(int i=0;i<v_steps;i++)
     {
      //--- use the following value
      if(p>1)
         p-=1;
      //--- move the point
      if(!ArrowRightPriceMove(0,InpName,date[d],price[p]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the label from the chart
   ArrowRightPriceDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_ARROW_BUY

Buy sign.



Example

The following script creates and moves Buy sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

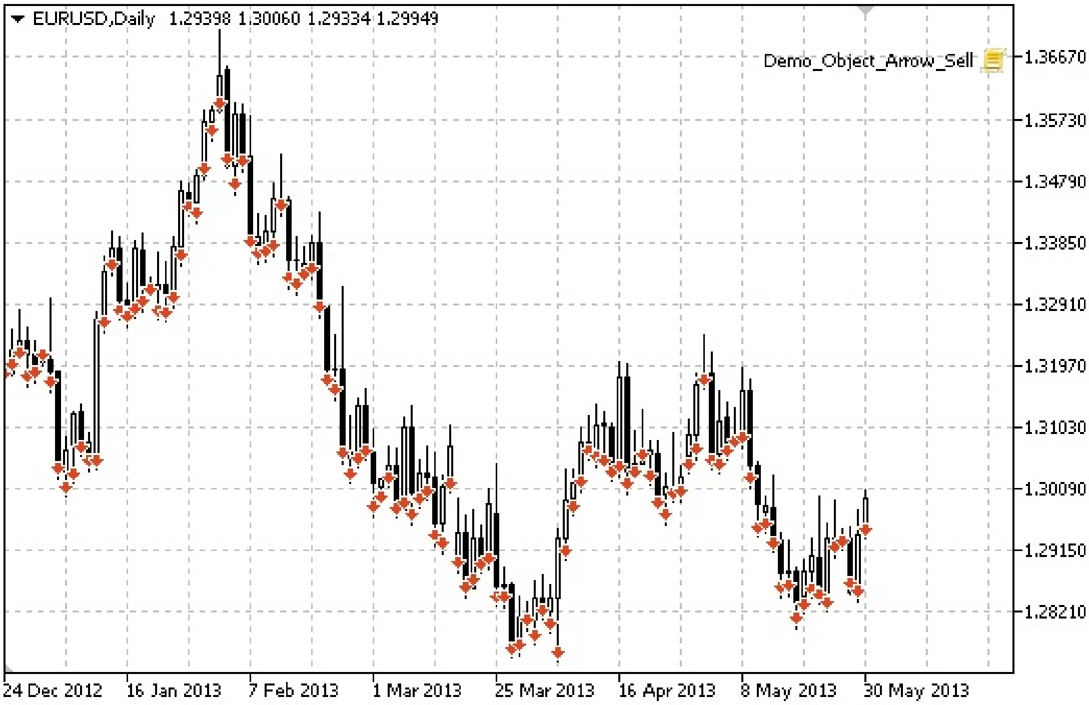
```
#property strict //--- description
#property description "Script draws \"Buy\" signs in the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input color InpColor=C'3,95,172'; // Color of signs
//+------------------------------------------------------------------+
//| Create Buy sign                                                  |
//+------------------------------------------------------------------+
bool ArrowBuyCreate(const long            chart_ID=0,        // chart's ID
                    const string          name="ArrowBuy",   // sign name
                    const int             sub_window=0,      // subwindow
                    datetime              time=0,            // anchor poi
                    double                price=0,           // anchor poi
                    const color           clr=C'3,95,172',   // sign colo
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // line style
                    const int                width=1,           // line size
                    const bool               back=false,        // in the bac
                    const bool               selection=false,   // highlight
                    const bool               hidden=true,       // hidden in
                    const long               z_order=0)         // priority f
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_BUY,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Buy\" sign! Error code = ",GetLastError(
      return(false);
     }
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set a line style (when highlighted)
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a line size (when highlighted)
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mous
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowBuyMove(const long   chart_ID=0,      // chart's ID
                  const string name="ArrowBuy", // object name
                  datetime     time=0,          // anchor point time coord
                  double       price=0)         // anchor point price coor
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
```

```
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Buy sign                                                  |
//+------------------------------------------------------------------+
bool ArrowBuyDelete(const long   chart_ID=0,       // chart's ID
                    const string name="ArrowBuy") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Buy\" sign! Error code = ",GetLastError(
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // array for storing dates of visible bars
   double   low[];  // array for storing Low prices of visible bars
   double   high[]; // array for storing High prices of visible bars
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of Low prices
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Failed to copy the values of Low prices! Error code = ",GetLa
      return;
     }
//--- fill the array of High prices
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Failed to copy the values of High prices! Error code = ",GetL
      return;
     }
//--- create Buy signs in Low point for each visible bar
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyCreate(0,"ArrowBuy_"+(string)i,0,date[i],low[i],InpColor
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- move Buy signs to High point for each visible bar
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyMove(0,"ArrowBuy_"+(string)i,date[i],high[i]))
```

```
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- delete Buy signs
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyDelete(0,"ArrowBuy_"+(string)i))
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//---
  }
```

OBJ_ARROW_SELL

Sell sign.



Example

The following script creates and moves Sell sign on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

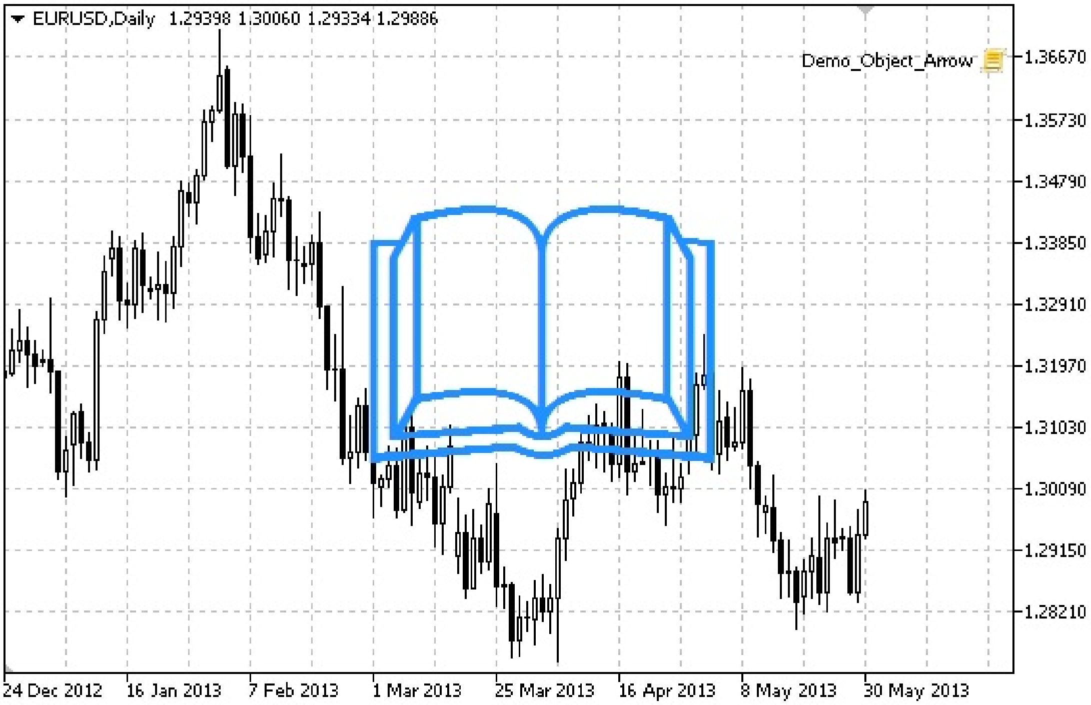
```
#property strict //--- description
#property description "Script draws \"Sell\" signs in the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input color InpColor=C'225,68,29'; // Color of signs
//+------------------------------------------------------------------+
//| Create Sell sign                                                 |
//+------------------------------------------------------------------+
bool ArrowSellCreate(const long            chart_ID=0,        // chart's I
                     const string          name="ArrowSell",  // sign name
                     const int             sub_window=0,      // subwindow
                     datetime              time=0,            // anchor po
                     double                price=0,           // anchor po
                     const color           clr=C'225,68,29',  // sign colo
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // line styl
                     const int                  width=1,            // line size
                     const bool                 back=false,         // in the ba
                     const bool                 selection=false,    // highlight
                     const bool                 hidden=true,        // hidden in
                     const long                 z_order=0)          // priority
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create the sign
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_SELL,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Sell\" sign! Error code = ",GetLastError
      return(false);
     }
//--- set a sign color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set a line style (when highlighted)
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a line size (when highlighted)
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the sign by mous
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowSellMove(const long   chart_ID=0,        // chart's ID
                   const string name="ArrowSell", // object name
                   datetime     time=0,            // anchor point time coo
                   double       price=0)           // anchor point price co
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
```

```
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Sell sign                                                 |
//+------------------------------------------------------------------+
bool ArrowSellDelete(const long   chart_ID=0,       // chart's ID
                     const string name="ArrowSell") // sign name
  {
//--- reset the error value
   ResetLastError();
//--- delete the sign
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Sell\" sign! Error code = ",GetLastError
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // array for storing dates of visible bars
   double   low[];  // array for storing Low prices of visible bars
   double   high[]; // array for storing High prices of visible bars
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of Low prices
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Failed to copy the values of Low prices! Error code = ",GetLa
      return;
     }
//--- fill the array of High prices
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Failed to copy the values of High prices! Error code = ",GetI
      return;
     }
//--- create Sell signs in High point for each visible bar
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellCreate(0,"ArrowSell_"+(string)i,0,date[i],high[i],InpCc
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- move Sell signs to Low point for each visible bar
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellMove(0,"ArrowSell_"+(string)i,date[i],low[i]))
```

```
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- delete Sell signs
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellDelete(0,"ArrowSell_"+(string)i))
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//---
  }
```

OBJ_ARROW

Arrow object.



Note

Anchor point position relative to the object can be selected from ENUM_ARROW_ANCHOR.

Large arrows (more than 5) can only be created by setting the appropriate OBJPROP_WIDTH property value when writing a code in MetaEditor.

The necessary arrow type can be selected by setting one of the Wingdings font's symbol codes.

Example

The following script creates Arrow object on the chart and changes its type. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

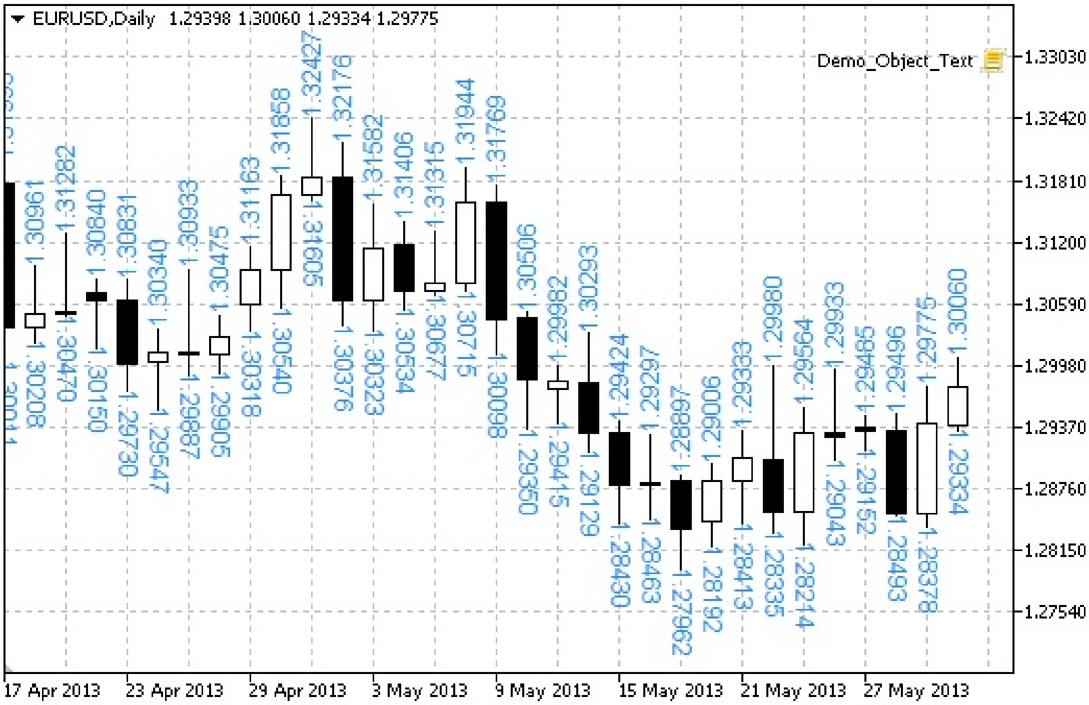
```
#property strict //--- description
#property description "Script creates a random arrow in the chart window."
#property description "Anchor point coordinate is set in"
#property description "percentage of the chart window size."
//--- display window of the input parameters during the script's launch
```

```
#property script_show_inputs
//--- input parameters of the script
input string            InpName="Arrow";        // Arrow name
input int               InpDate=50;             // Anchor point date in %
input int               InpPrice=50;            // Anchor point price in %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP;   // Anchor type
input color             InpColor=clrDodgerBlue; // Arrow color
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID;   // Border line style
input int               InpWidth=10;            // Arrow size
input bool              InpBack=false;          // Background arrow
input bool              InpSelection=false;     // Highlight to move
input bool              InpHidden=true;         // Hidden in the object li
input long              InpZOrder=0;            // Priority for mouse clic
//+------------------------------------------------------------------+
//| Create the arrow                                                 |
//+------------------------------------------------------------------+
bool ArrowCreate(const long              chart_ID=0,           // chart's
                 const string            name="Arrow",         // arrow na
                 const int               sub_window=0,         // subwindo
                 datetime                time=0,               // anchor p
                 double                  price=0,              // anchor p
                 const uchar             arrow_code=252,       // arrow co
                 const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // anchor p
                 const color             clr=clrRed,           // arrow co
                 const ENUM_LINE_STYLE   style=STYLE_SOLID,    // border l
                 const int               width=3,              // arrow si
                 const bool              back=false,           // in the b
                 const bool              selection=true,       // highligh
                 const bool              hidden=true,          // hidden i
                 const long              z_order=0)            // priority
  {
//--- set anchor point coordinates if they are not set
   ChangeArrowEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create an arrow
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create an arrow! Error code = ",GetLastError());
      return(false);
     }
//--- set the arrow code
   ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,arrow_code);
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set the arrow color
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set the arrow's size
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the arrow by mou
//--- when creating a graphical object using ObjectCreate function, the ob
//--- highlighted and moved by default. Inside this method, selection para
//--- is true by default making it possible to highlight and move the obje
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool ArrowMove(const long   chart_ID=0,   // chart's ID
               const string name="Arrow", // object name
               datetime     time=0,       // anchor point time coordinate
               double       price=0)      // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change the arrow code                                            |
//+------------------------------------------------------------------+
bool ArrowCodeChange(const long   chart_ID=0,    // chart's ID
                     const string name="Arrow", // object name
                     const uchar  code=252)      // arrow code
  {
//--- reset the error value
   ResetLastError();
//--- change the arrow code
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,code))
     {
      Print(__FUNCTION__,
            ": failed to change the arrow code! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change anchor type                                               |
//+------------------------------------------------------------------+
bool ArrowAnchorChange(const long              chart_ID=0,        // chart
                       const string            name="Arrow",      // objec
                       const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // ancho
  {
//--- reset the error value
   ResetLastError();
//--- change anchor type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": failed to change anchor type! Error code = ",GetLastError()
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete an arrow                                                  |
//+------------------------------------------------------------------+
bool ArrowDelete(const long   chart_ID=0,    // chart's ID
                 const string name="Arrow") // arrow name
  {
//--- reset the error value
   ResetLastError();
//--- delete an arrow
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete an arrow! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- price array size
   int accuracy=1000;
//--- arrays for storing the date and price values to be used
//--- for setting and changing sign anchor point coordinates
   datetime date[];
   double   price[];
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of prices
//--- find the highest and lowest values of the chart
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- define a change step of a price and fill the array
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- define points for drawing the arrow
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- create an arrow on the chart
   if(!ArrowCreate(0,InpName,0,date[d],price[p],32,InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart
   ChartRedraw();
//--- consider all cases of creating arrows in the loop
   for(int i=33;i<256;i++)
     {
      if(!ArrowCodeChange(0,InpName,(uchar)i))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // half a second of delay
      Sleep(500);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the arrow from the chart
   ArrowDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_TEXT

Text object.



Note

Anchor point position relative to the text can be selected from ENUM_ANCHOR_POINT enumeration. You can also change text slope angle using OBJPROP_ANGLE property.

Example

The following script creates several Text objects on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates \"Text\" graphical object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpFont="Arial";         // Font
input int               InpFontSize=10;          // Font size
input color             InpColor=clrRed;         // Color
input double            InpAngle=90.0;           // Slope angle in degrees
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_BOTTOM; // Anchor type
input bool              InpBack=false;          // Background object
input bool              InpSelection=false;     // Highlight to move
input bool              InpHidden=true;         // Hidden in the object l
input long              InpZOrder=0;            // Priority for mouse cli
//+------------------------------------------------------------------+
//| Creating Text object                                               |
//+------------------------------------------------------------------+
bool TextCreate(const long              chart_ID=0,               // chart
                const string            name="Text",              // objec
                const int               sub_window=0,             // subwi
                datetime                time=0,                   // ancho
                double                  price=0,                  // ancho
                const string            text="Text",              // the t
                const string            font="Arial",             // font
                const int               font_size=10,             // font
                const color             clr=clrRed,               // color
                const double            angle=0.0,                // text
                const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // ancho
                const bool              back=false,               // in th
                const bool              selection=false,          // highl
                const bool              hidden=true,              // hidde
                const long              z_order=0)                // prior
  {
//--- set anchor point coordinates if they are not set
   ChangeTextEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create Text object
   if(!ObjectCreate(chart_ID,name,OBJ_TEXT,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create \"Text\" object! Error code = ",GetLastErr
      return(false);
     }
//--- set the text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- set font size
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set the slope angle of the text
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- display in the foreground (false) or background (true)
```

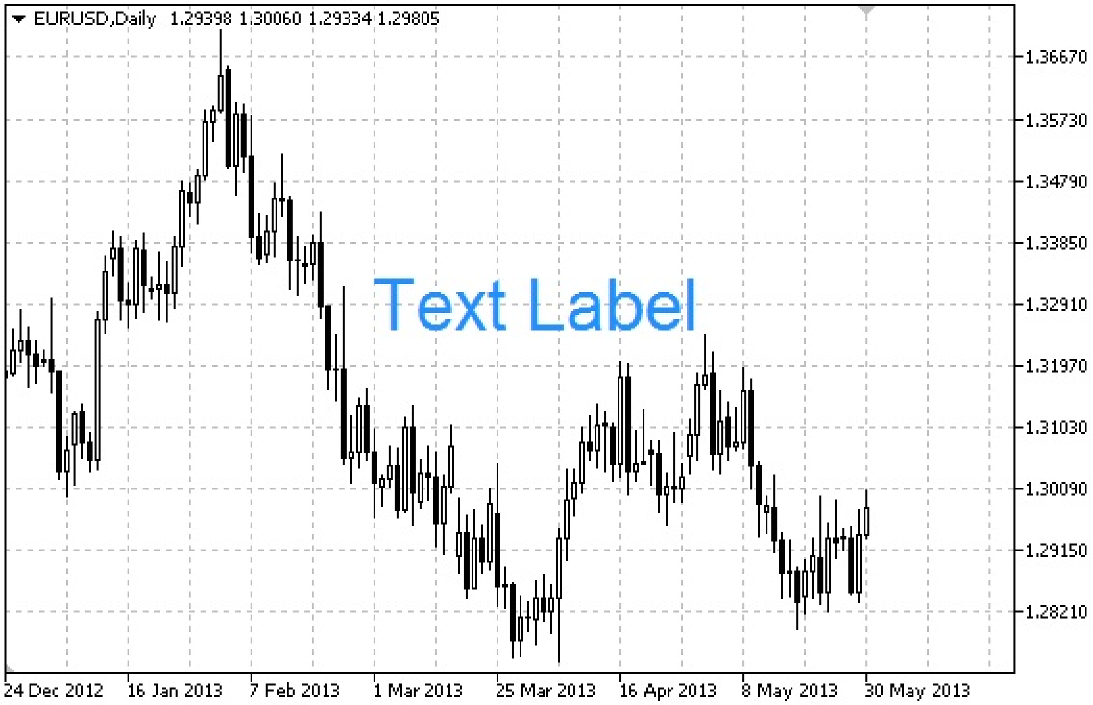
```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the object by mo
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the anchor point                                            |
//+------------------------------------------------------------------+
bool TextMove(const long   chart_ID=0,  // chart's ID
              const string name="Text", // object name
              datetime     time=0,      // anchor point time coordinate
              double       price=0)     // anchor point price coordinate
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change the object text                                           |
//+------------------------------------------------------------------+
bool TextChange(const long   chart_ID=0,  // chart's ID
                const string name="Text", // object name
                const string text="Text") // text
  {
//--- reset the error value
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
```

```
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Text object                                               |
//+------------------------------------------------------------------+
bool TextDelete(const long   chart_ID=0,  // chart's ID
                const string name="Text") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Text\" object! Error code = ",GetLastErr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeTextEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // array for storing dates of visible bars
   double   low[];  // array for storing Low prices of visible bars
   double   high[]; // array for storing High prices of visible bars
```

```
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of Low prices
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Failed to copy the values of Low prices! Error code = ",GetLa
      return;
     }
//--- fill the array of High prices
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Failed to copy the values of High prices! Error code = ",GetI
      return;
     }
//--- define how often texts are to be displayed
   int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- define the step
   int step=1;
   switch(scale)
     {
      case 0:
         step=12;
         break;
      case 1:
         step=6;
         break;
      case 2:
         step=4;
         break;
      case 3:
         step=2;
         break;
     }
//--- create texts for High and Low bars' values (with gaps)
   for(int i=0;i<bars;i+=step)
     {
      //--- create the texts
      if(!TextCreate(0,"TextHigh_"+(string)i,0,date[i],high[i],DoubleToStr
         InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOr
        {
         return;
        }
      if(!TextCreate(0,"TextLow_"+(string)i,0,date[i],low[i],DoubleToStrin
         InpColor,-InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZO
        {
         return;
        }
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- half a second of delay
   Sleep(500);
//--- delete the texts
   for(int i=0;i<bars;i+=step)
     {
      if(!TextDelete(0,"TextHigh_"+(string)i))
         return;
      if(!TextDelete(0,"TextLow_"+(string)i))
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//---
  }
```

OBJ_LABEL

Label object.



Note

For OBJ_LABEL, OBJ_BITMAP_LABEL and OBJ_RECTANGLE_LABEL, you can set the chart corner, relative to which the object anchor point is positioned. The corner is set using the OBJPROP_CORNER object property which can take one of the four values of ENUM_BASE_CORNER:

ID	Description
CORNER_LEFT_UPPER	Anchor point coordinates are set relative to the upper left corner of the chart
CORNER_LEFT_LOWER	Anchor point coordinates are set relative to the lower left corner of the chart
CORNER_RIGHT_LOWER	Anchor point coordinates are set relative to the lower right corner of the chart
CORNER_RIGHT_UPPER	Anchor point coordinates are set relative to the upper right corner of the chart

The position of the anchor points is set using the OBJPROP_ANCHOR property. It can be one of the 9 values of ENUM_ANCHOR_POINT:

ID	Description
ANCHOR_LEFT_UPPER	Anchor point in the upper left corner

ANCHOR_LEFT	Anchor point at the center of the left side
ANCHOR_LEFT_LOWER	Anchor point in the lower left corner
ANCHOR_LOWER	Anchor point at the center of the bottom side
ANCHOR_RIGHT_LOWER	Anchor point in the lower right corner
ANCHOR_RIGHT	Anchor point at the center of the right side
ANCHOR_RIGHT_UPPER	Anchor point in the upper right corner
ANCHOR_UPPER	Anchor point at the center of the top side
ANCHOR_CENTER	Anchor point at the center of the object

Example

The following script creates and moves Edit object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates \"Label\" graphical object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="Label";         // Label name
input int               InpX=150;                // X-axis distance
input int               InpY=150;                // Y-axis distance
input string            InpFont="Arial";         // Font
input int               InpFontSize=14;          // Font size
input color             InpColor=clrRed;         // Color
input double            InpAngle=0.0;            // Slope angle in degrees
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER; // Anchor type
input bool              InpBack=false;           // Background object
input bool              InpSelection=true;       // Highlight to move
input bool              InpHidden=true;          // Hidden in the object l
input long              InpZOrder=0;             // Priority for mouse cli
//+------------------------------------------------------------------+
//| Create a text label                                              |
//+------------------------------------------------------------------+
bool LabelCreate(const long              chart_ID=0,               // char
                 const string            name="Label",             // labe
                 const int               sub_window=0,             // subw
                 const int               x=0,                      // X co
                 const int               y=0,                      // Y co
                 const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // char
                 const string            text="Label",             // text
                 const string            font="Arial",             // font
                 const int               font_size=10,             // font
                 const color             clr=clrRed,               // colo
                 const double            angle=0.0,                // text
                 const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // anch
                 const bool              back=false,               // in t
                 const bool              selection=false,          // high
                 const bool              hidden=true,              // hidd
                 const long              z_order=0)                // pric
  {
//--- reset the error value
   ResetLastError();
//--- create a text label
   if(!ObjectCreate(chart_ID,name,OBJ_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create text label! Error code = ",GetLastError())
      return(false);
     }
//--- set label coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set the chart's corner, relative to which point coordinates are defi
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set the text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- set font size
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set the slope angle of the text
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mou
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
```

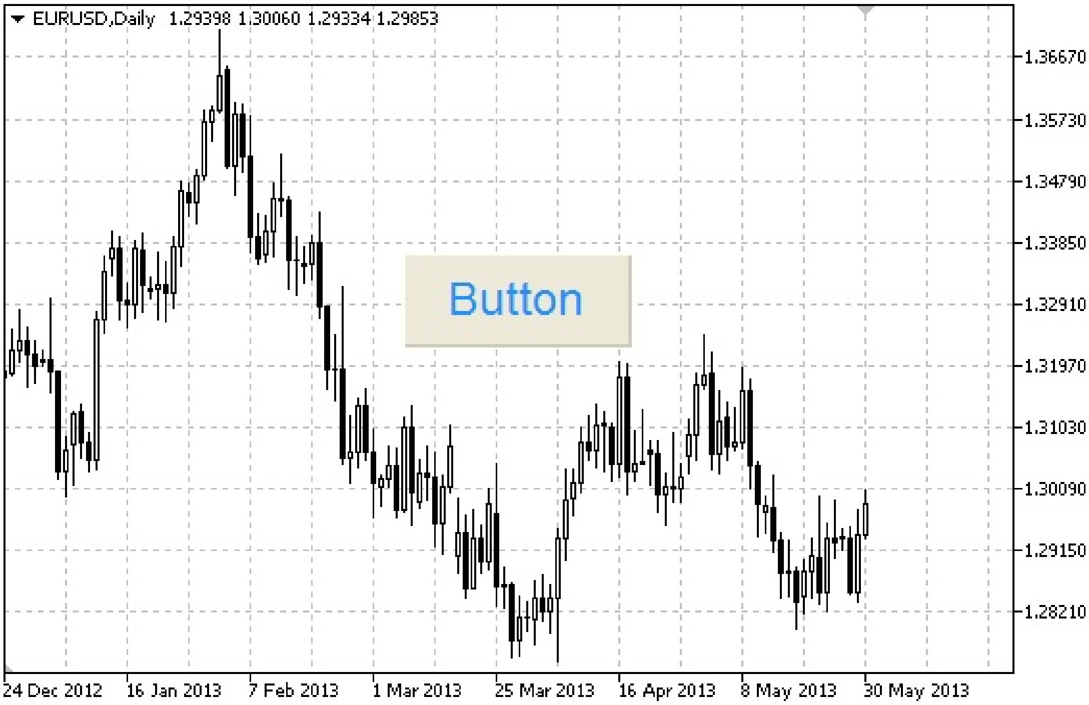
```
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the text label                                              |
//+------------------------------------------------------------------+
bool LabelMove(const long   chart_ID=0,   // chart's ID
               const string name="Label", // label name
               const int    x=0,          // X coordinate
               const int    y=0)          // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the text label
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the label! Error code = ",Ge
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the label! Error code = ",Ge
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change corner of the chart for binding the label                 |
//+------------------------------------------------------------------+
bool LabelChangeCorner(const long             chart_ID=0,               //
                       const string           name="Label",             //
                       const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) //
  {
//--- reset the error value
   ResetLastError();
//--- change anchor corner
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
     {
      Print(__FUNCTION__,
            ": failed to change the anchor corner! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
```

```
//+------------------------------------------------------------------+
//| Change the object text                                           |
//+------------------------------------------------------------------+
bool LabelTextChange(const long   chart_ID=0,   // chart's ID
                     const string name="Label", // object name
                     const string text="Text")  // text
  {
//--- reset the error value
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete a text label                                              |
//+------------------------------------------------------------------+
bool LabelDelete(const long   chart_ID=0,   // chart's ID
                 const string name="Label") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a text label! Error code = ",GetLastError(
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- store the label's coordinates in the local variables
   int x=InpX;
   int y=InpY;
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError())
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError()
      return;
     }
//--- check correctness of the input parameters
   if(InpX<0 || InpX>x_distance-1 || InpY<0 || InpY>y_distance-1)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- prepare initial text for the label
   string text;
   text=StringConcatenate("Upper left corner: ",x,",",y);
//--- create a text label on the chart
   if(!LabelCreate(0,InpName,0,InpX,InpY,CORNER_LEFT_UPPER,text,InpFont,In
      InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder
     {
      return;
     }
//--- redraw the chart and wait for half a second
   ChartRedraw();
   Sleep(500);
//--- move the label and change its text simultaneously
//--- number of iterations by axes
   int h_steps=(int)(x_distance/2-InpX);
   int v_steps=(int)(y_distance/2-InpY);
//--- move the label down
   for(int i=0;i<v_steps;i++)
     {
      //--- change the coordinate
      y+=2;
      //--- move the label and change its text
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- half a second of delay
   Sleep(500);
//--- move the label to the right
   for(int i=0;i<h_steps;i++)
     {
      //--- change the coordinate
      x+=2;
      //--- move the label and change its text
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- half a second of delay
   Sleep(500);
//--- move the label up
   for(int i=0;i<v_steps;i++)
     {
      //--- change the coordinate
      y-=2;
      //--- move the label and change its text
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- half a second of delay
   Sleep(500);
//--- move the label to the left
   for(int i=0;i<h_steps;i++)
     {
      //--- change the coordinate
      x-=2;
      //--- move the label and change its text
      MoveAndTextChange(x,y,"Upper left corner: ");
     }
//--- half a second of delay
   Sleep(500);
//--- now, move the point by changing the anchor corner
//--- move to the lower left corner
   if(!LabelChangeCorner(0,InpName,CORNER_LEFT_LOWER))
      return;
//--- change the label text
   text=StringConcatenate("Lower left corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- redraw the chart and wait for two seconds
   ChartRedraw();
   Sleep(2000);
//--- move to the lower right corner
   if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_LOWER))
      return;
//--- change the label text
   text=StringConcatenate("Lower right corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- redraw the chart and wait for two seconds
   ChartRedraw();
   Sleep(2000);
//--- move to the upper right corner
   if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_UPPER))
      return;
//--- change the label text
   text=StringConcatenate("Upper right corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- redraw the chart and wait for two seconds
   ChartRedraw();
   Sleep(2000);
//--- move to the upper left corner
   if(!LabelChangeCorner(0,InpName,CORNER_LEFT_UPPER))
      return;
//--- change the label text
   text=StringConcatenate("Upper left corner: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- redraw the chart and wait for two seconds
   ChartRedraw();
   Sleep(2000);
//--- delete the label
   LabelDelete(0,InpName);
//--- redraw the chart and wait for half a second
   ChartRedraw();
   Sleep(500);
//---
  }
//+------------------------------------------------------------------+
//| The function moves the object and changes its text               |
//+------------------------------------------------------------------+
bool MoveAndTextChange(const int x,const int y,string text)
  {
//--- move the label
   if(!LabelMove(0,InpName,x,y))
      return(false);
//--- change the label text
   text=StringConcatenate(text,x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return(false);
//--- check if the script's operation has been forcefully disabled
   if(IsStopped())
      return(false);
//--- redraw the chart
   ChartRedraw();
// 0.01 seconds of delay
   Sleep(10);
```

```
//--- exit the function
   return(true);
  }
```

OBJ_BUTTON

Button object.



Note

Anchor point coordinates are set in pixels. You can select button anchoring corner from ENUM_BASE_CORNER.

Example

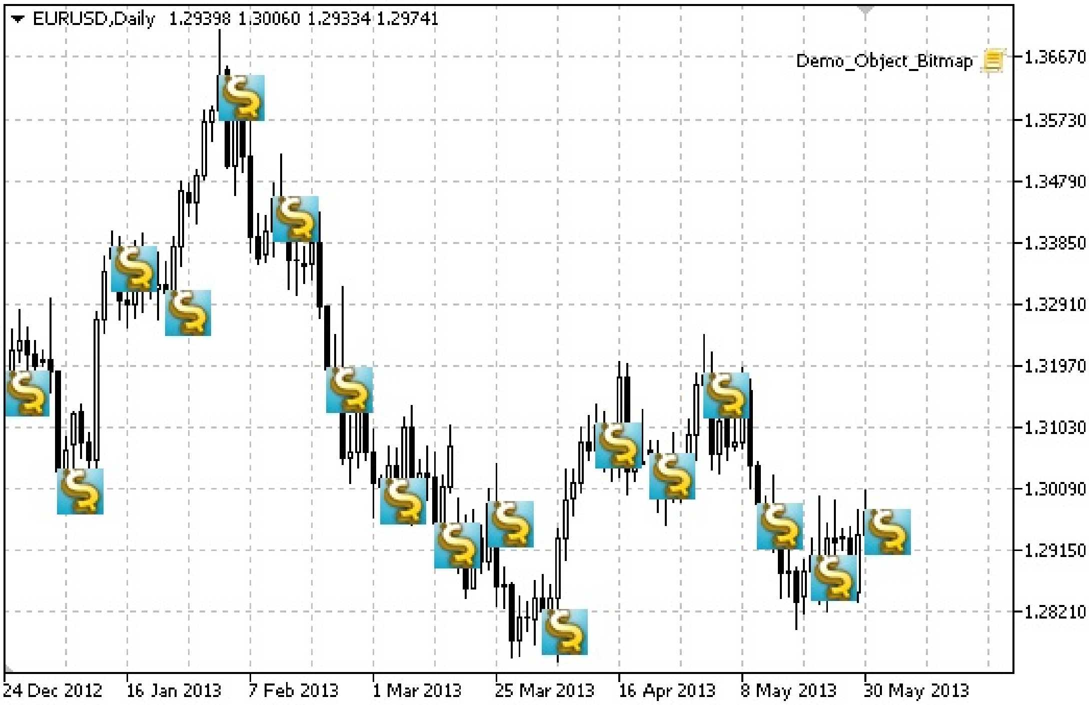
The following script creates and moves Button object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates the button on the chart."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string           InpName="Button";            // Button name
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for an
input string           InpFont="Arial";             // Font
input int              InpFontSize=14;              // Font size
input color            InpColor=clrBlack;           // Text color
input color            InpBackColor=C'236,233,216'; // Background color
input color            InpBorderColor=clrNONE;      // Border color
input bool             InpState=false;              // Pressed/Released
input bool             InpBack=false;               // Background object
input bool             InpSelection=false;          // Highlight to move
input bool             InpHidden=true;              // Hidden in the objec
input long             InpZOrder=0;                 // Priority for mouse
//+------------------------------------------------------------------+
//| Create the button                                                |
//+------------------------------------------------------------------+
bool ButtonCreate(const long              chart_ID=0,               // cha
                  const string            name="Button",            // but
                  const int               sub_window=0,             // sub
                  const int               x=0,                      // X c
                  const int               y=0,                      // Y c
                  const int               width=50,                 // but
                  const int               height=18,                // but
                  const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // cha
                  const string            text="Button",            // tex
                  const string            font="Arial",             // fon
                  const int               font_size=10,             // fon
                  const color             clr=clrBlack,             // tex
                  const color             back_clr=C'236,233,216',  // bac
                  const color             border_clr=clrNONE,       // bor
                  const bool              state=false,              // pre
                  const bool              back=false,               // in
                  const bool              selection=false,          // hig
                  const bool              hidden=true,              // hid
                  const long              z_order=0)                // pri
  {
//--- reset the error value
   ResetLastError();
//--- create the button
   if(!ObjectCreate(chart_ID,name,OBJ_BUTTON,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create the button! Error code = ",GetLastError())
      return(false);
     }
//--- set button coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set button size
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the chart's corner, relative to which point coordinates are defi
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set the text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
```

```
//--- set font size
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set text color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set background color
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- set border color
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- set button state
   ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);
//--- enable (true) or disable (false) the mode of moving the button by mo
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move the button                                                  |
//+------------------------------------------------------------------+
bool ButtonMove(const long   chart_ID=0,    // chart's ID
                const string name="Button", // button name
                const int    x=0,           // X coordinate
                const int    y=0)           // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the button
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the button! Error code = ",G
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the button! Error code = ",G
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change button size                                               |
//+------------------------------------------------------------------+
bool ButtonChangeSize(const long   chart_ID=0,    // chart's ID
                      const string name="Button", // button name
                      const int    width=50,      // button width
                      const int    height=18)     // button height
  {
//--- reset the error value
   ResetLastError();
//--- change the button size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the button width! Error code = ",GetLastEr
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the button height! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change corner of the chart for binding the button                |
//+------------------------------------------------------------------+
bool ButtonChangeCorner(const long             chart_ID=0,               /
                        const string           name="Button",            /
                        const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) /
  {
//--- reset the error value
   ResetLastError();
//--- change anchor corner
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
     {
      Print(__FUNCTION__,
            ": failed to change the anchor corner! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change button text                                               |
//+------------------------------------------------------------------+
bool ButtonTextChange(const long   chart_ID=0,    // chart's ID
                      const string name="Button", // button name
                      const string text="Text")   // text
  {
//--- reset the error value
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the button                                                |
//+------------------------------------------------------------------+
bool ButtonDelete(const long   chart_ID=0,    // chart's ID
                  const string name="Button") // button name
  {
//--- reset the error value
   ResetLastError();
//--- delete the button
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete the button! Error code = ",GetLastError())
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError())
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError()
      return;
     }
//--- define the step for changing the button size
   int x_step=(int)x_distance/32;
   int y_step=(int)y_distance/32;
//--- set the button coordinates and its size
   int x=(int)x_distance/32;
   int y=(int)y_distance/32;
   int x_size=(int)x_distance*15/16;
   int y_size=(int)y_distance*15/16;
//--- create the button
   if(!ButtonCreate(0,InpName,0,x,y,x_size,y_size,InpCorner,"Press",InpFon
      InpColor,InpBackColor,InpBorderColor,InpState,InpBack,InpSelection,I
     {
      return;
     }
//--- redraw the chart
   ChartRedraw();
//--- reduce the button in the loop
   int i=0;
   while(i<13)
     {
      //--- half a second of delay
      Sleep(500);
      //--- switch the button to the pressed state
      ObjectSetInteger(0,InpName,OBJPROP_STATE,true);
      //--- redraw the chart and wait for 0.2 second
      ChartRedraw();
      Sleep(200);
      //--- redefine coordinates and button size
      x+=x_step;
      y+=y_step;
      x_size-=x_step*2;
      y_size-=y_step*2;
      //--- reduce the button
      ButtonMove(0,InpName,x,y);
      ButtonChangeSize(0,InpName,x_size,y_size);
      //--- bring the button back to the released state
      ObjectSetInteger(0,InpName,OBJPROP_STATE,false);
      //--- redraw the chart
      ChartRedraw();
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- increase the loop counter
      i++;
     }
//--- half a second of delay
   Sleep(500);
//--- delete the button
   ButtonDelete(0,InpName);
   ChartRedraw();
//--- wait for 1 second
   Sleep(1000);
//---
  }
```

OBJ_BITMAP

Bitmap object.



Note

For Bitmap object, you can select visibility scope of an image.

Example

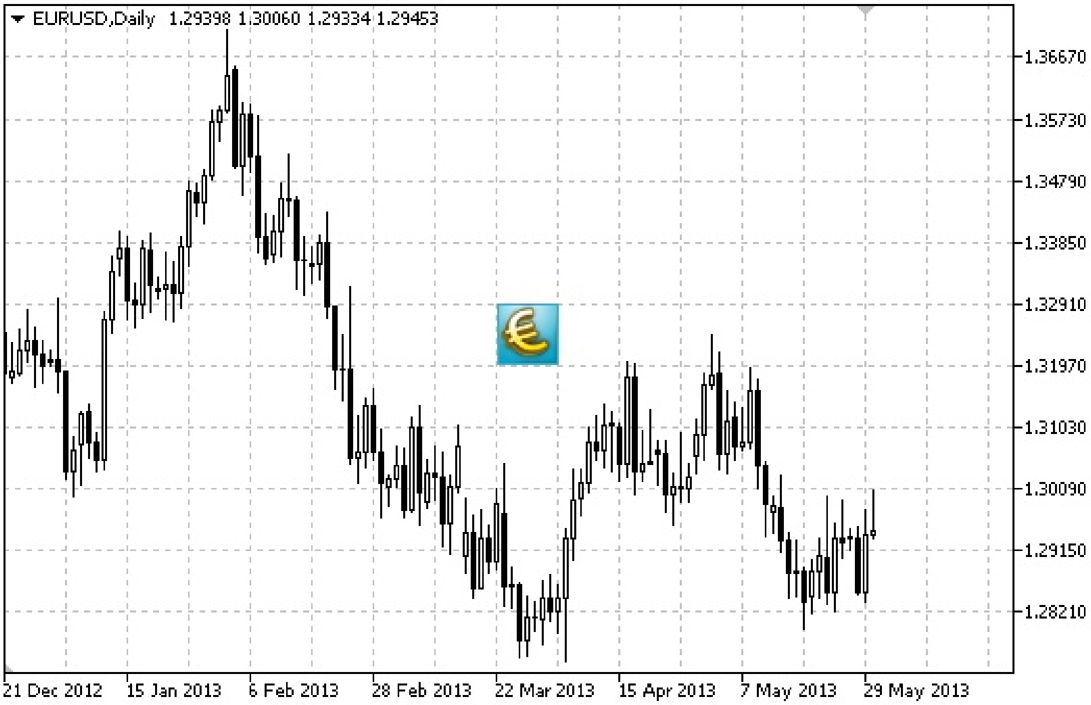
The following script creates several bitmaps on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates a bitmap in the chart window."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpFile="\\Images\\dollar.bmp"; // Bitmap file name
input int             InpWidth=24;                    // Visibility scope
input int             InpHeight=24;                   // Visibility scope
input int             InpXOffset=4;                   // Visibility scope
input int             InpYOffset=4;                   // Visibility scope
input color           InpColor=clrRed;                // Border color wher
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID;           // Line style when h
input int             InpPointWidth=1;                // Point size to mov
input bool            InpBack=false;                  // Background object
input bool            InpSelection=false;             // Highlight to move
input bool            InpHidden=true;                 // Hidden in the obj
input long            InpZOrder=0;                    // Priority for mous
//+------------------------------------------------------------------+
//| Create a bitmap in the chart window                              |
//+------------------------------------------------------------------+
bool BitmapCreate(const long            chart_ID=0,        // chart's ID
                  const string          name="Bitmap",     // bitmap name
                  const int             sub_window=0,      // subwindow in
                  datetime              time=0,            // anchor point
                  double                price=0,           // anchor point
                  const string          file="",           // bitmap file
                  const int             width=10,          // visibility s
                  const int             height=10,         // visibility s
                  const int             x_offset=0,        // visibility s
                  const int             y_offset=0,        // visibility s
                  const color           clr=clrRed,        // border color
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // line style w
                  const int             point_width=1,     // move point s
                  const bool            back=false,        // in the backg
                  const bool            selection=false,   // highlight to
                  const bool            hidden=true,       // hidden in th
                  const long            z_order=0)         // priority for
  {
//--- set anchor point coordinates if they are not set
   ChangeBitmapEmptyPoint(time,price);
//--- reset the error value
   ResetLastError();
//--- create a bitmap
   if(!ObjectCreate(chart_ID,name,OBJ_BITMAP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": failed to create a bitmap in the chart window! Error code =
      return(false);
     }
//--- set the path to the image file
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
     {
      Print(__FUNCTION__,
            ": failed to load the image! Error code = ",GetLastError());
      return(false);
     }
//--- set visibility scope for the image; if width or height values
//--- exceed the width and height (respectively) of a source image,
//--- it is not drawn; in the opposite case,
//--- only the part corresponding to these values is drawn
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the part of an image that is to be displayed in the visibility s
//--- the default part is the upper left area of an image; the values allo
//--- performing a shift from this area displaying another part of the ima
   ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
   ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- set the border color when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a size of the anchor point for moving an object
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mou
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set a new image for the bitmap                                   |
//+------------------------------------------------------------------+
bool BitmapSetImage(const long   chart_ID=0,    // chart's ID
                    const string name="Bitmap", // bitmap name
                    const string file="")       // path to the file
  {
//--- reset the error value
   ResetLastError();
//--- set the path to the image file
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
     {
      Print(__FUNCTION__,
            ": failed to load the image! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move a bitmap in the chart window                                |
//+------------------------------------------------------------------+
bool BitmapMove(const long   chart_ID=0,     // chart's ID
                const string name="Bitmap", // bitmap name
                datetime     time=0,         // anchor point time
                double       price=0)        // anchor point price
  {
//--- if point position is not set, move it to the current bar having Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- reset the error value
   ResetLastError();
//--- move the anchor point
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": failed to move the anchor point! Error code = ",GetLastErro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change visibility scope (bitmap) size                            |
//+------------------------------------------------------------------+
bool BitmapChangeSize(const long   chart_ID=0,     // chart's ID
                      const string name="Bitmap", // bitmap name
                      const int    width=0,        // bitmap width
                      const int    height=0)       // bitmap height
  {
//--- reset the error value
   ResetLastError();
//--- change bitmap size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the bitmap width! Error code = ",GetLastEr
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the bitmap height! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
```

```
   return(true);
  }
//+----------------------------------------------------------------------+
//| Change coordinate of the upper left corner of the visibility scope |
//+----------------------------------------------------------------------+
bool BitmapMoveVisibleArea(const long   chart_ID=0,    // chart's ID
                           const string name="Bitmap", // bitmap name
                           const int    x_offset=0,    // visibility scope
                           const int    y_offset=0)    // visibility scope
  {
//--- reset the error value
   ResetLastError();
//--- change the bitmap's visibility scope coordinates
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
     {
      Print(__FUNCTION__,
            ": failed to change X coordinate of the visibility scope! Erro
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
     {
      Print(__FUNCTION__,
            ": failed to change Y coordinate of the visibility scope! Erro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete a bitmap                                                  |
//+------------------------------------------------------------------+
bool BitmapDelete(const long   chart_ID=0,    // chart's ID
                  const string name="Bitmap") // bitmap name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a bitmap! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Check anchor point values and set default values                 |
//| for empty ones                                                   |
//+------------------------------------------------------------------+
void ChangeBitmapEmptyPoint(datetime &time,double &price)
  {
//--- if the point's time is not set, it will be on the current bar
   if(!time)
      time=TimeCurrent();
//--- if the point's price is not set, it will have Bid value
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[];  // array for storing dates of visible bars
   double   close[]; // array for storing Close prices
//--- bitmap file name
   string   file="\\Images\\dollar.bmp";
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- memory allocation
   ArrayResize(date,bars);
   ArrayResize(close,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- fill the array of Close prices
   if(CopyClose(Symbol(),Period(),0,bars,close)==-1)
     {
      Print("Failed to copy the values of Close prices! Error code = ",Get
      return;
     }
//--- define how often the images should be displayed
   int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- define the step
   int step=1;
   switch(scale)
     {
      case 0:
         step=27;
         break;
      case 1:
         step=14;
         break;
      case 2:
         step=7;
         break;
      case 3:
         step=4;
         break;
      case 4:
         step=2;
         break;
     }
//--- create bitmaps for High and Low bars' values (with gaps)
   for(int i=0;i<bars;i+=step)
     {
      //--- create the bitmaps
      if(!BitmapCreate(0,"Bitmap_"+(string)i,0,date[i],close[i],InpFile,In
         InpYOffset,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,I
        {
         return;
        }
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- half a second of delay
   Sleep(500);
//--- delete Sell signs
   for(int i=0;i<bars;i+=step)
     {
      if(!BitmapDelete(0,"Bitmap_"+(string)i))
         return;
      if(!BitmapDelete(0,"Bitmap_"+(string)i))
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//---
  }
```

OBJ_BITMAP_LABEL

Bitmap Label object.



Note

For bitmap label, you can select visibility scope of an image.

For OBJ_LABEL, OBJ_BITMAP_LABEL and OBJ_RECTANGLE_LABEL, you can set the chart corner, relative to which the object anchor point is positioned. The corner is set using the OBJPROP_CORNER object property which can take one of the four values of ENUM_BASE_CORNER:

ID	Description
CORNER_LEFT_UPPER	Anchor point coordinates are set relative to the upper left of the chart
CORNER_LEFT_LOWER	Anchor point coordinates are set relative to the lower heft corner of the chart
CORNER_RIGHT_LOWER	Anchor point coordinates are set relative to the lower right corner of the chart
CORNER_RIGHT_UPPER	Anchor point coordinates are set relative to the upper right corner of the chart

The position of the anchor points is set using the OBJPROP_ANCHOR property. It can be one of the 9 values of ENUM_ANCHOR_POINT:

ID	Description

ANCHOR_LEFT_UPPER	Anchor point in the upper left corner
ANCHOR_LEFT	Anchor point at the center of the left side
ANCHOR_LEFT_LOWER	Anchor point in the lower left corner
ANCHOR_LOWER	Anchor point at the center of the bottom side
ANCHOR_RIGHT_LOWER	Anchor point in the lower right corner
ANCHOR_RIGHT	Anchor point at the center of the right side
ANCHOR_RIGHT_UPPER	Anchor point in the upper right corner
ANCHOR_UPPER	Anchor point at the center of the top side
ANCHOR_CENTER	Anchor point at the center of the object

Example

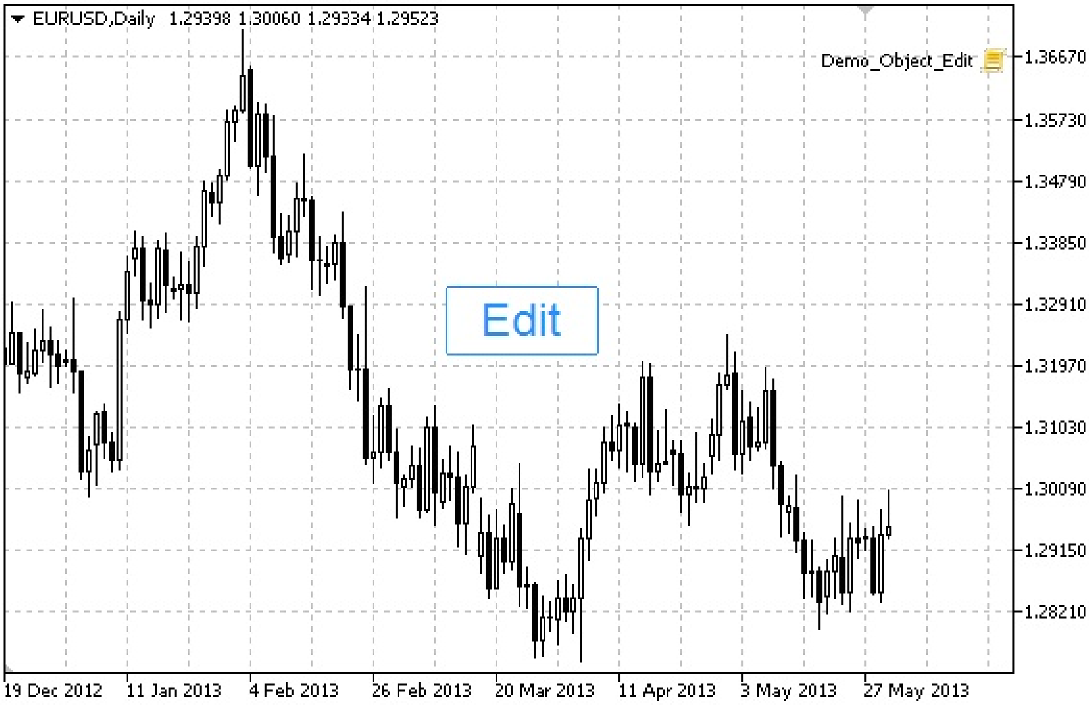
The following script creates several bitmaps on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates \"Bitmap Label\" object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string            InpName="BmpLabel";                // Label name
input string            InpFileOn="\\Images\\dollar.bmp"; // File name for
input string            InpFileOff="\\Images\\euro.bmp";  // File name for
input bool              InpState=false;                    // Label pressed
input ENUM_BASE_CORNER  InpCorner=CORNER_LEFT_UPPER;       // Chart corner
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER;           // Anchor type
input color             InpColor=clrRed;                   // Border color
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID;              // Line style wh
input int               InpPointWidth=1;                   // Point size to
input bool              InpBack=false;                     // Background ob
input bool              InpSelection=false;                // Highlight to
input bool              InpHidden=true;                    // Hidden in the
input long              InpZOrder=0;                       // Priority for
//+------------------------------------------------------------------+
//| Create Bitmap Label object                                       |
//+------------------------------------------------------------------+
bool BitmapLabelCreate(const long              chart_ID=0,               /
                       const string            name="BmpLabel",          /
                       const int               sub_window=0,             /
                       const int               x=0,                      /
                       const int               y=0,                       /
                       const string            file_on="",                /
                       const string            file_off="",               /
                       const int               width=0,                   /
                       const int               height=0,                  /
                       const int               x_offset=10,               /
                       const int               y_offset=10,               /
                       const bool              state=false,               /
                       const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, /
                       const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, /
                       const color             clr=clrRed,                /
                       const ENUM_LINE_STYLE   style=STYLE_SOLID,         /
                       const int               point_width=1,             /
                       const bool              back=false,                /
                       const bool              selection=false,           /
                       const bool              hidden=true,               /
                       const long              z_order=0)                 /
  {
//--- reset the error value
   ResetLastError();
//--- create a bitmap label
   if(!ObjectCreate(chart_ID,name,OBJ_BITMAP_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Bitmap Label\" object! Error code = ",Ge
      return(false);
     }
//--- set the images for On and Off modes
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,0,file_on))
     {
      Print(__FUNCTION__,
            ": failed to load the image for On mode! Error code = ",GetLas
      return(false);
     }
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,1,file_off))
     {
      Print(__FUNCTION__,
            ": failed to load the image for Off mode! Error code = ",GetLa
      return(false);
     }
//--- set label coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set visibility scope for the image; if width or height values
//--- exceed the width and height (respectively) of a source image,
//--- it is not drawn; in the opposite case,
//--- only the part corresponding to these values is drawn
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the part of an image that is to be displayed in the visibility s
//--- the default part is the upper left area of an image; the values allo
//--- performing a shift from this area displaying another part of the ima
   ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
   ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- define the label's status (pressed or released)
   ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);
//--- set the chart's corner, relative to which point coordinates are defi
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set anchor type
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- set the border color when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set the border line style when object highlighting mode is enabled
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set a size of the anchor point for moving an object
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mou
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Set a new image for Bitmap label object                          |
//+------------------------------------------------------------------+
bool BitmapLabelSetImage(const long   chart_ID=0,      // chart's ID
                         const string name="BmpLabel", // label name
                         const int    on_off=0,        // modifier (On or
                         const string file="")         // path to the file
  {
//--- reset the error value
   ResetLastError();
//--- set the path to the image file
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,on_off,file))
     {
      Print(__FUNCTION__,
            ": failed to load the image! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Bitmap Label object                                         |
//+------------------------------------------------------------------+
bool BitmapLabelMove(const long   chart_ID=0,      // chart's ID
                     const string name="BmpLabel", // label name
                     const int    x=0,             // X coordinate
                     const int    y=0)             // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the object
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the object! Error code = ",G
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the object! Error code = ",G
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change visibility scope (object) size                            |
//+------------------------------------------------------------------+
bool BitmapLabelChangeSize(const long   chart_ID=0,      // chart's ID
                           const string name="BmpLabel", // label name
                           const int    width=0,         // label width
                           const int    height=0)        // label height
  {
//--- reset the error value
   ResetLastError();
//--- change the object size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the object width! Error code = ",GetLastEr
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the object height! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+---------------------------------------------------------------------+
//| Change coordinate of the upper left corner of the visibility scope |
//+---------------------------------------------------------------------+
bool BitmapLabelMoveVisibleArea(const long   chart_ID=0,      // chart's I
                                const string name="BmpLabel", // label nam
                                const int    x_offset=0,      // visibilit
                                const int    y_offset=0)      // visibilit
  {
//--- reset the error value
   ResetLastError();
//--- change the object's visibility scope coordinates
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
     {
      Print(__FUNCTION__,
            ": failed to change X coordinate of the visibility scope! Erro
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
     {
      Print(__FUNCTION__,
            ": failed to change Y coordinate of the visibility scope! Erro
      return(false);
     }
//--- successful execution
   return(true);
  }
//+-------------------------------------------------------------------+
//| Delete "Bitmap label" object                                      |
//+-------------------------------------------------------------------+
bool BitmapLabelDelete(const long   chart_ID=0,      // chart's ID
                       const string name="BmpLabel") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Bitmap label\" object! Error code = ",Ge
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError())
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError()
      return;
     }
//--- define bitmap label coordinates
   int x=(int)x_distance/2;
   int y=(int)y_distance/2;
//--- set label size and visibility scope coordinates
   int width=32;
   int height=32;
   int x_offset=0;
   int y_offset=0;
//--- place bitmap label at the center of the window
   if(!BitmapLabelCreate(0,InpName,0,x,y,InpFileOn,InpFileOff,width,height
      InpCorner,InpAnchor,InpColor,InpStyle,InpPointWidth,InpBack,InpSelec
     {
      return;
     }
//--- redraw the chart and wait one second
   ChartRedraw();
   Sleep(1000);
//--- change label's visibility scope size in the loop
   for(int i=0;i<6;i++)
     {
      //--- change visibility scope size
      width--;
      height--;
      if(!BitmapLabelChangeSize(0,InpName,width,height))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.3 seconds of delay
      Sleep(300);
     }
//--- 1 second of delay
   Sleep(1000);
//--- change label's visibility scope coordinates in the loop
   for(int i=0;i<2;i++)
     {
      //--- change visibility scope coordinates
      x_offset++;
      y_offset++;
      if(!BitmapLabelMoveVisibleArea(0,InpName,x_offset,y_offset))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.3 seconds of delay
      Sleep(300);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the label
   BitmapLabelDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_EDIT

Edit object.



Note

Anchor point coordinates are set in pixels. You can select Edit anchoring corner from ENUM_BASE_CORNER enumeration.

You can also select one of the text alignment types inside Edit from ENUM_ALIGN_MODE enumeration.

Example

The following script creates and moves Edit object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates \"Edit\" object."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string           InpName="Edit";              // Object name
input string           InpText="Text";              // Object text
input string           InpFont="Arial";             // Font
input int              InpFontSize=14;              // Font size
input ENUM_ALIGN_MODE  InpAlign=ALIGN_CENTER;       // Text alignment type
input bool             InpReadOnly=false;           // Ability to edit
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for an
input color            InpColor=clrBlack;           // Text color
input color            InpBackColor=clrWhite;       // Background color
input color            InpBorderColor=clrBlack;     // Border color
input bool             InpBack=false;               // Background object
input bool             InpSelection=false;          // Highlight to move
input bool             InpHidden=true;              // Hidden in the objec
input long             InpZOrder=0;                 // Priority for mouse
//+------------------------------------------------------------------+
//| Create Edit object                                               |
//+------------------------------------------------------------------+
bool EditCreate(const long             chart_ID=0,               // chart'
                const string           name="Edit",              // object
                const int              sub_window=0,             // subwin
                const int              x=0,                      // X coor
                const int              y=0,                      // Y coor
                const int              width=50,                 // width
                const int              height=18,                // height
                const string           text="Text",              // text
                const string           font="Arial",             // font
                const int              font_size=10,             // font s
                const ENUM_ALIGN_MODE  align=ALIGN_CENTER,       // alignm
                const bool             read_only=false,          // abilit
                const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // chart
                const color            clr=clrBlack,             // text c
                const color            back_clr=clrWhite,        // backgr
                const color            border_clr=clrNONE,       // border
                const bool             back=false,               // in the
                const bool             selection=false,          // highli
                const bool             hidden=true,              // hidden
                const long             z_order=0)                // priori
  {
//--- reset the error value
   ResetLastError();
//--- create edit field
   if(!ObjectCreate(chart_ID,name,OBJ_EDIT,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Edit\" object! Error code = ",GetLastErr
      return(false);
     }
//--- set object coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set object size
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set the text
```

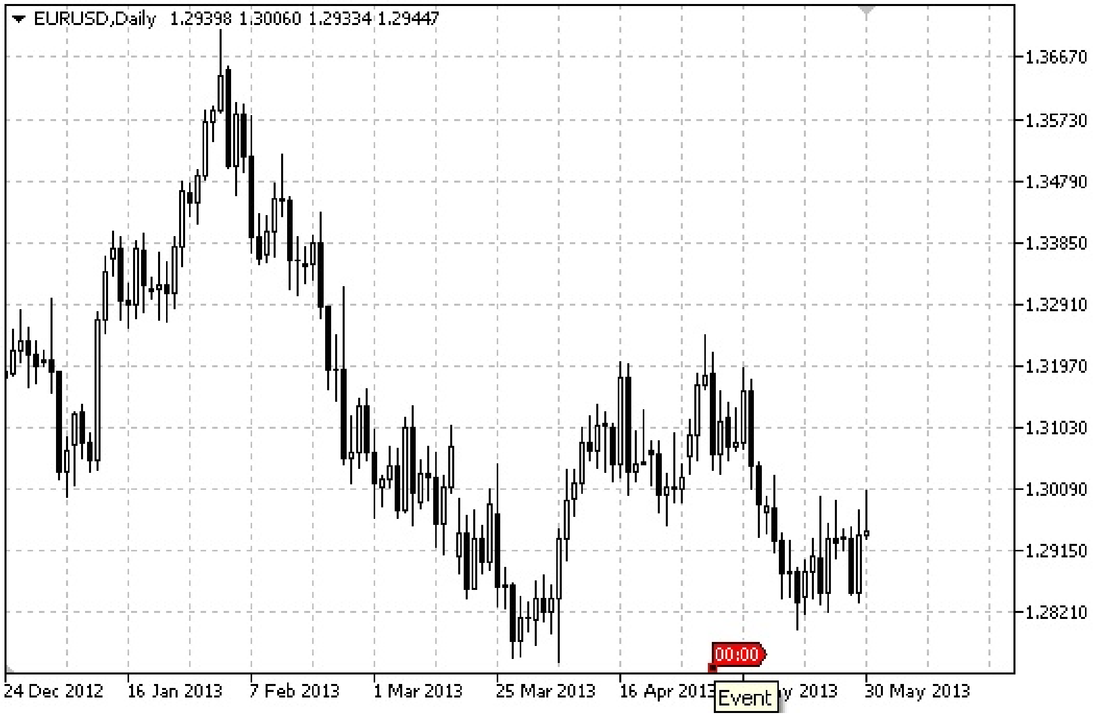
```
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set text font
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- set font size
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- set the type of text alignment in the object
   ObjectSetInteger(chart_ID,name,OBJPROP_ALIGN,align);
//--- enable (true) or cancel (false) read-only mode
   ObjectSetInteger(chart_ID,name,OBJPROP_READONLY,read_only);
//--- set the chart's corner, relative to which object coordinates are def
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set text color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set background color
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- set border color
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mou
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Edit object                                                 |
//+------------------------------------------------------------------+
bool EditMove(const long   chart_ID=0,  // chart's ID
              const string name="Edit", // object name
              const int    x=0,         // X coordinate
              const int    y=0)         // Y coordinate
  {
//--- reset the error value
   ResetLastError();
//--- move the object
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the object! Error code = ",G
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
```

```
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the object! Error code = ",G
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Resize Edit object                                               |
//+------------------------------------------------------------------+
bool EditChangeSize(const long   chart_ID=0,  // chart's ID
                    const string name="Edit", // object name
                    const int    width=0,     // width
                    const int    height=0)    // height
  {
//--- reset the error value
   ResetLastError();
//--- change the object size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the object width! Error code = ",GetLastEr
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the object height! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Edit object's text                                        |
//+------------------------------------------------------------------+
bool EditTextChange(const long   chart_ID=0,  // chart's ID
                    const string name="Edit", // object name
                    const string text="Text") // text
  {
//--- reset the error value
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Return Edit object text                                          |
//+------------------------------------------------------------------+
bool EditTextGet(string      &text,        // text
                 const long   chart_ID=0,  // chart's ID
                 const string name="Edit") // object name
  {
//--- reset the error value
   ResetLastError();
//--- get object text
   if(!ObjectGetString(chart_ID,name,OBJPROP_TEXT,0,text))
     {
      Print(__FUNCTION__,
            ": failed to get the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Edit object                                               |
//+------------------------------------------------------------------+
bool EditDelete(const long   chart_ID=0,  // chart's ID
                const string name="Edit") // object name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Edit\" object! Error code = ",GetLastErr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
```

```
  {
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError())
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Failed to get the chart height! Error code = ",GetLastError()
      return;
     }
//--- define the step for changing the edit field
   int x_step=(int)x_distance/64;
//--- set edit field coordinates and its size
   int x=(int)x_distance/8;
   int y=(int)y_distance/2;
   int x_size=(int)x_distance/8;
   int y_size=InpFontSize*2;
//--- store the text in the local variable
   string text=InpText;
//--- create edit field
   if(!EditCreate(0,InpName,0,x,y,x_size,y_size,InpText,InpFont,InpFontSiz
      InpCorner,InpColor,InpBackColor,InpBorderColor,InpBack,InpSelection,
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- stretch the edit field
   while(x_size-x<x_distance*5/8)
     {
      //--- increase edit field's width
      x_size+=x_step;
      if(!EditChangeSize(0,InpName,x_size,y_size))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart and wait for 0.05 seconds
      ChartRedraw();
      Sleep(50);
     }
//--- half a second of delay
   Sleep(500);
//--- change the text
   for(int i=0;i<20;i++)
     {
      //--- add "+" at the beginning and at the end
      text="+"+text+"+";
      if(!EditTextChange(0,InpName,text))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart and wait for 0.1 seconds
      ChartRedraw();
      Sleep(100);
     }
//--- half a second of delay
   Sleep(500);
//--- delete edit field
   EditDelete(0,InpName);
   ChartRedraw();
//--- wait for 1 second
   Sleep(1000);
//---
  }
```

OBJ_EVENT

Event object.



Note

When hovering mouse over the event, its text appears.

Example

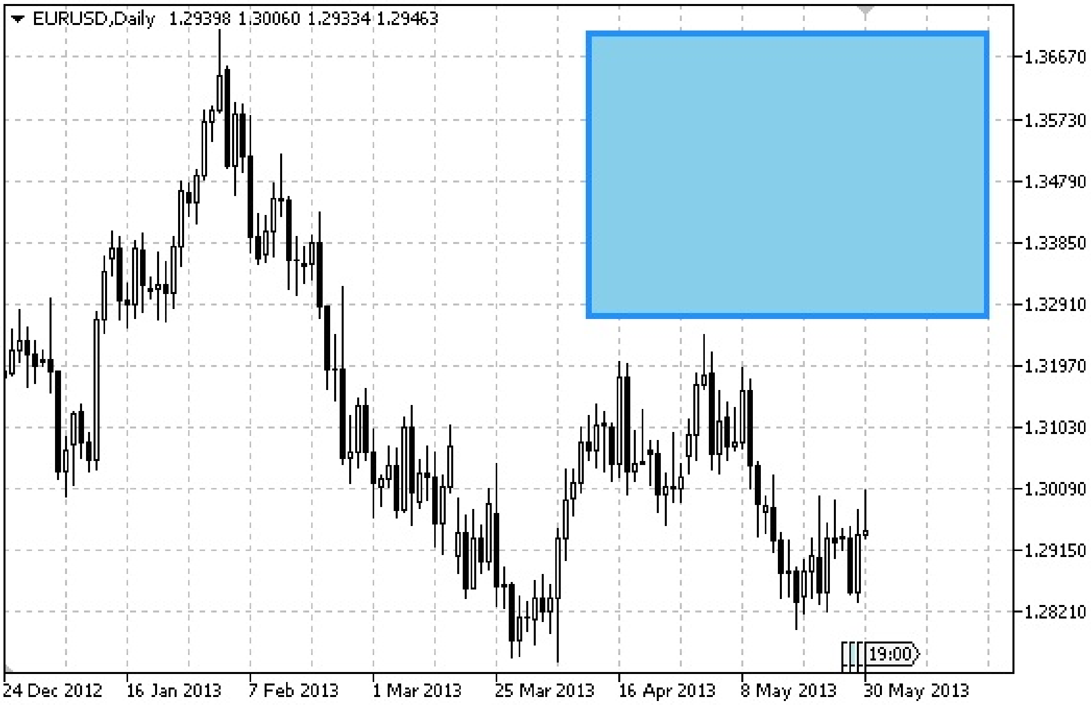
The following script creates and moves Event object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script draws \"Event\" graphical object."
#property description "Anchor point date is set in percentage of"
#property description "the chart window width in bars."
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string          InpName="Event";    // Event name
input int             InpDate=25;         // Event date, %
input string          InpText="Text";     // Event text
input color           InpColor=clrRed;    // Event color
input int             InpWidth=1;         // Point size when highlighted
input bool            InpBack=false;      // Background event
input bool            InpSelection=false; // Highlight to move
input bool            InpHidden=true;     // Hidden in the object list
input long            InpZOrder=0;        // Priority for mouse click
//+------------------------------------------------------------------+
//| Create Event object on the chart                                 |
//+------------------------------------------------------------------+
bool EventCreate(const long            chart_ID=0,      // chart's ID
                 const string          name="Event",    // event name
                 const int             sub_window=0,    // subwindow index
                 const string          text="Text",     // event text
                 datetime              time=0,          // time
                 const color           clr=clrRed,      // color
                 const int             width=1,         // point width whe
                 const bool            back=false,      // in the backgrou
                 const bool            selection=false, // highlight to mo
                 const bool            hidden=true,     // hidden in the c
                 const long            z_order=0)       // priority for mo
  {
//--- if time is not set, create the object on the last bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- create Event object
   if(!ObjectCreate(chart_ID,name,OBJ_EVENT,sub_window,time,0))
     {
      Print(__FUNCTION__,
            ": failed to create \"Event\" object! Error code = ",GetLastEr
      return(false);
     }
//--- set event text
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- set color
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set anchor point width if the object is highlighted
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving event by mouse
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Change Event object text                                         |
//+------------------------------------------------------------------+
bool EventTextChange(const long   chart_ID=0,   // chart's ID
                     const string name="Event", // event name
                     const string text="Text")  // text
  {
//--- reset the error value
   ResetLastError();
//--- change object text
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": failed to change the text! Error code = ",GetLastError());
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move Event object                                                |
//+------------------------------------------------------------------+
bool EventMove(const long   chart_ID=0,   // chart's ID
               const string name="Event", // event name
               datetime     time=0)       // time
  {
//--- if time is not set, move event to the last bar
   if(!time)
      time=TimeCurrent();
//--- reset the error value
   ResetLastError();
//--- move the object
   if(!ObjectMove(chart_ID,name,0,time,0))
     {
      Print(__FUNCTION__,
            ": failed to move \"Event\" object! Error code = ",GetLastErrc
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete Event object                                              |
//+------------------------------------------------------------------+
bool EventDelete(const long   chart_ID=0,   // chart's ID
                 const string name="Event") // event name
  {
//--- reset the error value
   ResetLastError();
//--- delete the object
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete \"Event\" object! Error code = ",GetLastEr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- check correctness of the input parameters
   if(InpDate<0 || InpDate>100)
     {
      Print("Error! Incorrect values of input parameters!");
      return;
     }
//--- number of visible bars in the chart window
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- array for storing the date values to be used
//--- for setting and changing line anchor point's coordinates
   datetime date[];
//--- memory allocation
   ArrayResize(date,bars);
//--- fill the array of dates
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Failed to copy time values! Error code = ",GetLastError());
      return;
     }
//--- define the points to create an object
   int d=InpDate*(bars-1)/100;
//--- create Event object
   if(!EventCreate(0,InpName,0,InpText,date[d],InpColor,InpWidth,
      InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- now, move the object
//--- loop counter
   int h_steps=bars/2;
//--- move the object
   for(int i=0;i<h_steps;i++)
     {
      //--- use the following value
      if(d<bars-1)
         d+=1;
      //--- move the point
      if(!EventMove(0,InpName,date[d]))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart
      ChartRedraw();
      // 0.05 seconds of delay
      Sleep(50);
     }
//--- 1 second of delay
   Sleep(1000);
//--- delete the channel from the chart
   EventDelete(0,InpName);
   ChartRedraw();
//--- 1 second of delay
   Sleep(1000);
//---
  }
```

OBJ_RECTANGLE_LABEL

Rectangle Label object.



Note

The function is used for creating and designing the graphical user interface. The frame type for the rectangular label can be selected from the enumeration ENUM_BORDER_TYPE.

For OBJ_LABEL, OBJ_BITMAP_LABEL and OBJ_RECTANGLE_LABEL, you can set the chart corner, relative to which the object anchor point is positioned. The corner is set using the OBJPROP_CORNER object property which can take one of the four values of ENUM_BASE_CORNER:

ID	Description
CORNER_LEFT_UPPER	Anchor point coordinates are set relative to the upper left corner of the chart
CORNER_LEFT_LOWER	Anchor point coordinates are set relative to the lower left corner of the chart
CORNER_RIGHT_LOWER	Anchor point coordinates are set relative to the lower right corner of the chart
CORNER_RIGHT_UPPER	Anchor point coordinates are set relative to the upper right corner of the chart

Example

The following script creates and moves Rectangle Label object on the chart. Special functions have been developed to create and change graphical object's properties. You can use these functions "as is" in your own applications.

```
#property strict //--- description
#property description "Script creates \"Rectangle Label\" graphical object
//--- display window of the input parameters during the script's launch
#property script_show_inputs
//--- input parameters of the script
input string           InpName="RectLabel";         // Label name
input color            InpBackColor=clrSkyBlue;     // Background color
input ENUM_BORDER_TYPE InpBorder=BORDER_FLAT;       // Border type
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Chart corner for an
input color            InpColor=clrDarkBlue;        // Flat border color (
input ENUM_LINE_STYLE  InpStyle=STYLE_SOLID;        // Flat border style (
input int              InpLineWidth=3;              // Flat border width (
input bool             InpBack=false;               // Background object
input bool             InpSelection=true;           // Highlight to move
input bool             InpHidden=true;              // Hidden in the objec
input long             InpZOrder=0;                 // Priority for mouse
//+------------------------------------------------------------------+
//| Create rectangle label                                           |
//+------------------------------------------------------------------+
bool RectLabelCreate(const long             chart_ID=0,               // c
                     const string           name="RectLabel",         // l
                     const int              sub_window=0,             // s
                     const int              x=0,                      // X
                     const int              y=0,                      // Y
                     const int              width=50,                 // w
                     const int              height=18,                // h
                     const color            back_clr=C'236,233,216',  // b
                     const ENUM_BORDER_TYPE border=BORDER_SUNKEN,     // b
                     const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // c
                     const color            clr=clrRed,               // f
                     const ENUM_LINE_STYLE  style=STYLE_SOLID,        // f
                     const int              line_width=1,             // f
                     const bool             back=false,               // i
                     const bool             selection=false,          // h
                     const bool             hidden=true,              // h
                     const long             z_order=0)                // p
  {
//--- reset the error value
   ResetLastError();
```

```
//--- create a rectangle label
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": failed to create a rectangle label! Error code = ",GetLastE
      return(false);
     }
//--- set label coordinates
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- set label size
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- set background color
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- set border type
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border);
//--- set the chart's corner, relative to which point coordinates are defi
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- set flat border color (in Flat mode)
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- set flat border line style
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- set flat border width
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,line_width);
//--- display in the foreground (false) or background (true)
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- enable (true) or disable (false) the mode of moving the label by mou
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- hide (true) or display (false) graphical object name in the object l
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- set the priority for receiving the event of a mouse click in the cha
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Move rectangle label                                             |
//+------------------------------------------------------------------+
bool RectLabelMove(const long   chart_ID=0,       // chart's ID
                   const string name="RectLabel", // label name
                   const int    x=0,              // X coordinate
                   const int    y=0)              // Y coordinate
  {
//--- reset the error value
   ResetLastError();
```

```
//--- move the rectangle label
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": failed to move X coordinate of the label! Error code = ",Ge
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": failed to move Y coordinate of the label! Error code = ",Ge
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change the size of the rectangle label                           |
//+------------------------------------------------------------------+
bool RectLabelChangeSize(const long   chart_ID=0,       // chart's ID
                         const string name="RectLabel", // label name
                         const int    width=50,         // label width
                         const int    height=18)        // label height
  {
//--- reset the error value
   ResetLastError();
//--- change label size
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": failed to change the label's width! Error code = ",GetLastE
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": failed to change the label's height! Error code = ",GetLast
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Change rectangle label border type                               |
//+------------------------------------------------------------------+
bool RectLabelChangeBorderType(const long             chart_ID=0,
                               const string           name="RectLabel",
                               const ENUM_BORDER_TYPE border=BORDER_SUNKEN
  {
//--- reset the error value
   ResetLastError();
//--- change border type
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border))
     {
      Print(__FUNCTION__,
            ": failed to change the border type! Error code = ",GetLastErr
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Delete the rectangle label                                       |
//+------------------------------------------------------------------+
bool RectLabelDelete(const long   chart_ID=0,       // chart's ID
                     const string name="RectLabel") // label name
  {
//--- reset the error value
   ResetLastError();
//--- delete the label
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": failed to delete a rectangle label! Error code = ",GetLastE
      return(false);
     }
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- chart window size
   long x_distance;
   long y_distance;
//--- set window size
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Failed to get the chart width! Error code = ",GetLastError())
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
```

```
     {
      Print("Failed to get the chart height! Error code = ",GetLastError()
      return;
     }
//--- define rectangle label coordinates
   int x=(int)x_distance/4;
   int y=(int)y_distance/4;
//--- set label size
   int width=(int)x_distance/4;
   int height=(int)y_distance/4;
//--- create a rectangle label
   if(!RectLabelCreate(0,InpName,0,x,y,width,height,InpBackColor,InpBorder
      InpColor,InpStyle,InpLineWidth,InpBack,InpSelection,InpHidden,InpZOr
     {
      return;
     }
//--- redraw the chart and wait one second
   ChartRedraw();
   Sleep(1000);
//--- change the size of the rectangle label
   int steps=(int)MathMin(x_distance/4,y_distance/4);
   for(int i=0;i<steps;i++)
     {
      //--- resize
      width+=1;
      height+=1;
      if(!RectLabelChangeSize(0,InpName,width,height))
         return;
      //--- check if the script's operation has been forcefully disabled
      if(IsStopped())
         return;
      //--- redraw the chart and wait for 0.01 seconds
      ChartRedraw();
      Sleep(10);
     }
//--- 1 second of delay
   Sleep(1000);
//--- change border type
   if(!RectLabelChangeBorderType(0,InpName,BORDER_RAISED))
      return;
//--- redraw the chart and wait for 1 second
   ChartRedraw();
   Sleep(1000);
//--- change border type
   if(!RectLabelChangeBorderType(0,InpName,BORDER_SUNKEN))
      return;
//--- redraw the chart and wait for 1 second
```

```
   ChartRedraw();
   Sleep(1000);
//--- delete the label
   RectLabelDelete(0,InpName);
   ChartRedraw();
//--- wait for 1 second
   Sleep(1000);
//---
  }
```

Object Properties

Graphical objects can have various properties depending on the object type. All objects used in technical analysis are bound to the time and price coordinates: trendline, channels, Fibonacci tools, etc. But there is a number of auxiliary objects intended to improve the user interface that are bound to the always visible part of a chart (main chart windows or indicator subwindows):

Object	ID	X/Y	Width/Height	Date/Price	OBJPROP_CC
Text	OBJ_TEXT			Yes	
Label	OBJ_LABEL	Yes	Yes (read only)		Yes
Button	OBJ_BUTTON	Yes	Yes		Yes
Bitmap	OBJ_BITMAP		Yes (read only)	Yes	
Bitmap Label	OBJ_BITMAP_LABEL	Yes	Yes (read only)		Yes
Edit	OBJ_EDIT	Yes	Yes		Yes
Rectangle Label	OBJ_RECTANGLE_LABEL	Yes	Yes		Yes

The following designations are used in the table:

- **X/Y** coordinates of anchor points specified in pixels relative to a chart corner;
- **Width/Height** objects have width and height. For "read only", the width and height values are calculated only once the object is rendered on chart;
- **Date/Price** anchor point coordinates are specified using the date and price values;
- **OBJPROP_CORNER** defines the chart corner relative to which the anchor point coordinates are specified. Can be one of the 4 values of the ENUM_BASE_CORNER enumeration;
- **OBJPROP_ANCHOR** defines the anchor point in object itself and can be one of the 9 values of the ENUM_ANCHOR_POINT enumeration. Coordinates in pixels are specified from this very point to selected chart corner;
- **OBJPROP_ANGLE** defines the object rotation angle counterclockwise.

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

ID	Value	Type	Description
OBJPROP_TIME1	0	datetime	Datetime value to set/get first coordinate time part
OBJPROP_PRICE1	1	double	Double value to set/get first coordinate price part
OBJPROP_TIME2	2	datetime	Datetime value to set/get second coordinate time part
OBJPROP_PRICE2	3	double	Double value to set/get second coordinate price part
OBJPROP_TIME3	4	datetime	Datetime value to set/get third coordinate time part
OBJPROP_PRICE3	5	double	Double value to set/get third coordinate price part
OBJPROP_COLOR	6	color	Color value to set/get object color
OBJPROP_STYLE	7	int	Value is one of STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT constants to set/get object line style
OBJPROP_WIDTH	8	int	Integer value to set/get object line width. Can be from 1 to 5
OBJPROP_BACK	9	bool	Boolean value to set/get background drawing flag for object
OBJPROP_RAY	10	bool	Boolean value to set/get ray flag of object.
OBJPROP_ELLIPSE	11	bool	Boolean value to set/get ellipse flag for fibo arcs
OBJPROP_SCALE	12	double	Double value to set/get scale object property
OBJPROP_ANGLE	13	double	Double value to set/get angle object property in degrees
OBJPROP_ARROWCODE	14	int	Integer value or arrow enumeration to set/get arrow code object property
OBJPROP_TIMEFRAMES	15	int	Value can be one or combination (bitwise addition) of object visibility constants to set/get timeframe object property

OBJPROP_DEVIATION	16	double	Double value to set/get deviation property for Standard deviation objects
OBJPROP_FONTSIZE	100	int	Integer value to set/get font size for text objects
OBJPROP_CORNER	101	int	Integer value to set/get anchor corner property for label objects. Must be from 0-3.
OBJPROP_XDISTANCE	102	int	Integer value to set/get anchor X distance object property in pixels (see note)
OBJPROP_YDISTANCE	103	int	Integer value is to set/get anchor Y distance object property in pixels (see note)
OBJPROP_FIBOLEVELS	200	int	Integer value to set/get Fibonacci object level count. Can be from 0 to 32
OBJPROP_LEVELCOLOR	201	color	Color value to set/get object level line color
OBJPROP_LEVELSTYLE	202	int	Value is one of STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT constants to set/get object level line style
OBJPROP_LEVELWIDTH	203	int	Integer value to set/get object level line width. Can be from 1 to 5
OBJPROP_FIRSTLEVEL+n	210+n	int	Integer value to set/get the value of Fibonacci object level with index n. Index n can be from 0 (number of levels -1), but not larger than 31

Every graphical object in a price chart has a certain set of properties. Values of object properties are set up and received by corresponding functions for working with graphical objects. For each object type there is its own set of properties. Here all possible values from the ENUM_OBJECT_PROPERTY enumeration family are listed. Some properties require clarification, such as the level number for the Fibonacci extension object. In such cases it is necessary to specify the value of the *modifier* parameter in the functions of ObjectSet...() and ObjectGet...().

For functions ObjectSetInteger() and ObjectGetInteger()

ENUM_OBJECT_PROPERTY_INTEGER

Identifier	Description	Property Type
OBJPROP_COLOR	Color	color
OBJPROP_STYLE	Style	ENUM_LINE_STYLE
OBJPROP_WIDTH	Line thickness	int
OBJPROP_BACK	Object in the background	bool
OBJPROP_ZORDER	Priority of a graphical object for receiving events of clicking on a chart (CHARTEVENT_CLICK). The default zero value is set when creating an object; the priority can be increased if necessary. When objects are placed one atop another, only one of them with the highest priority will receive the CHARTEVENT_CLICK event.	long
OBJPROP_HIDDEN	Prohibit showing of the name of a graphical object in the list of objects from the terminal menu "Charts" - "Objects" - "List of objects". The true value allows to hide an object from the list. By default, true is set to the objects that display calendar events, trading history and to the objects created from MQL4 programs. To see such graphical objects and	bool

	access their properties, click on the "All" button in the "List of objects" window.	
OBJPROP_SELECTED	Object is selected	bool
OBJPROP_READONLY	Ability to edit text in the Edit object	bool
OBJPROP_TYPE	Object type	ENUM_OBJECT r/o
OBJPROP_TIME	Time coordinate	datetime modifier=number of anchor point
OBJPROP_SELECTABLE	Object availability	bool
OBJPROP_CREATETIME	Time of object creation	datetime r/o
OBJPROP_LEVELS	Number of levels	int
OBJPROP_LEVELCOLOR	Color of the line-level	color modifier=level number
OBJPROP_LEVELSTYLE	Style of the line-level	ENUM_LINE_STYLE modifier=level number
OBJPROP_LEVELWIDTH	Thickness of the line-level	int modifier=level number
OBJPROP_ALIGN	Horizontal text alignment in the "Edit" object (OBJ_EDIT)	ENUM_ALIGN_MODE
OBJPROP_FONTSIZE	Font size	int
OBJPROP_RAY_RIGHT	Ray goes to the right	bool
OBJPROP_ELLIPSE	Showing the full ellipse of the Fibonacci Arc object (OBJ_FIBOARC)	bool
OBJPROP_ARROWCODE	Arrow code for the Arrow object	char
OBJPROP_TIMEFRAMES	Visibility of an object at timeframes	set of flags flags
OBJPROP_ANCHOR	Location of the anchor point of a graphical object	ENUM_ARROW_ANCHOR (for OBJ_ARROW), ENUM_ANCHOR_POINT (for OBJ_LABEL, OBJ_BITMAP_LABEL and

		OBJ_TEXT)
OBJPROP_XDISTANCE	The distance in pixels along the X axis from the binding corner (see note)	int
OBJPROP_YDISTANCE	The distance in pixels along the Y axis from the binding corner (see note)	int
OBJPROP_DRAWLINES	Displaying lines for marking the Elliott Wave	bool
OBJPROP_STATE	Button state (pressed / depressed)	bool
OBJPROP_XSIZE	The object's width along the X axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	int
OBJPROP_YSIZE	The object's height along the Y axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	int
OBJPROP_XOFFSET	The X coordinate of the upper left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap"	int

	(OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	
OBJPROP_YOFFSET	The Y coordinate of the upper left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	int
OBJPROP_BGCOLOR	The background color for OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL	color
OBJPROP_CORNER	The corner of the chart to link a graphical object	ENUM_BASE_CORNER
OBJPROP_BORDER_TYPE	Border type for the "Rectangle label" object	ENUM_BORDER_TYPE
OBJPROP_BORDER_COLOR	Border color for the OBJ_EDIT and OBJ_BUTTON objects	color

For objects OBJ_BITMAP_LABEL and OBJ_BITMAP, a special mode of image display can be set programmatically. In this mode, only part of an original image (at which a rectangular visible area is applied) is displayed, while the rest of the image becomes invisible. The size of this area should be set using the properties OBJPROP_XSIZE and OBJPROP_YSIZE. The visible area can be "moved" only within the original image using the properties OBJPROP_XOFFSET and OBJPROP_YOFFSET.

For the fixed-sized objects: OBJ_BUTTON, OBJ_RECTANGLE_LABEL and OBJ_EDIT, properties OBJPROP_XDISTANCE and OBJPROP_YDISTANCE set the position of the top left point of the object relative to the chart corner (OBJPROP_CORNER), from which the X and Y coordinates will be counted in pixels.

For functions ObjectSetDouble() and ObjectGetDouble()

ENUM_OBJECT_PROPERTY_DOUBLE

Identifier	Description	Property Type
OBJPROP_PRICE	Price coordinate	double modifier=number of anchor point
OBJPROP_LEVELVALUE	Level value	double modifier=level number
OBJPROP_SCALE	Scale (properties of Gann objects, Fibonacci Arcs and Ellipse)	double
OBJPROP_ANGLE	Angle. For the objects with no angle specified, created from a program, the value is equal to EMPTY_VALUE	double
OBJPROP_DEVIATION	Deviation for the Standard Deviation Channel	double

For functions ObjectSetString() and ObjectGetString()

ENUM_OBJECT_PROPERTY_STRING

Identifier	Description	Property Type
OBJPROP_NAME	Object name	string
OBJPROP_TEXT	Description of the object (the text contained in the object)	string
OBJPROP_TOOLTIP	The text of a tooltip. If the property is not set, then the tooltip generated automatically by the terminal is shown. A tooltip can be disabled by setting the "\n" (line	string

	feed) value to it	
OBJPROP_LEVELTEXT	Level description	string modifier=level number
OBJPROP_FONT	Font	string
OBJPROP_BMPFILE	The name of BMP-file for Bitmap Label. See also Resources	string modifier: 0-state ON, 1-state OFF
OBJPROP_SYMBOL	Symbol for the Chart object	string

For the OBJ_RECTANGLE_LABEL object ("Rectangle label") one of the three design modes can be set, to which the following values of ENUM_BORDER_TYPE correspond.

ENUM_BORDER_TYPE

Identifier	Description
BORDER_FLAT	Flat form
BORDER_RAISED	Prominent form
BORDER_SUNKEN	Concave form

For the OBJ_EDIT object ("Edit") and for the ChartScreenShot()function, you can specify the horizontal alignment type using the values of the ENUM_ALIGN_MODE enumeration.

ENUM_ALIGN_MODE

Identifier	Description
ALIGN_LEFT	Left alignment
ALIGN_CENTER	Centered (only for the Edit object)
ALIGN_RIGHT	Right alignment

Example:

```
#define  UP          "\x0431" //+------------------------------------------
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   string label_name="my_OBJ_LABEL_object";
   if(ObjectFind(0,label_name)<0)
     {
      Print("Object ",label_name," not found. Error code = ",GetLastError(
      //--- create Label object
      ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
      //--- set X coordinate
      ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
      //--- set Y coordinate
      ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
      //--- define text color
      ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrWhite);
      //--- define text for object Label
      ObjectSetString(0,label_name,OBJPROP_TEXT,UP);
      //--- define font
      ObjectSetString(0,label_name,OBJPROP_FONT,"Wingdings");
      //--- define font size
      ObjectSetInteger(0,label_name,OBJPROP_FONTSIZE,10);
      //--- 45 degrees rotation clockwise
      ObjectSetDouble(0,label_name,OBJPROP_ANGLE,-45);
      //--- disable for mouse selecting
      ObjectSetInteger(0,label_name,OBJPROP_SELECTABLE,false);
      //--- draw it on the chart
      ChartRedraw(0);
     }
  }
```

Methods of Object Binding

Graphical objects Text, Label, Bitmap and Bitmap Label (OBJ_TEXT, OBJ_LABEL, OBJ_BITMAP and OBJ_BITMAP_LABEL) can have one of the 9 different ways of coordinate binding defined by the OBJPROP_ANCHOR property.

Object	ID	X/Y	Width/Height	Date/Price	OBJPROP_CC
Text	OBJ_TEXT			Yes	
Label	OBJ_LABEL	Yes	Yes (read only)		Yes
Button	OBJ_BUTTON	Yes	Yes		Yes
Bitmap	OBJ_BITMAP		Yes (read only)	Yes	
Bitmap Label	OBJ_BITMAP_LABEL	Yes	Yes (read only)		Yes
Edit	OBJ_EDIT	Yes	Yes		Yes
Rectangle Label	OBJ_RECTANGLE_LABEL	Yes	Yes		Yes

The following designations are used in the table:

- **X/Y** coordinates of anchor points specified in pixels relative to a chart corner;
- **Width/Height** objects have width and height. For "read only", the width and height values are calculated only once the object is rendered on chart;
- **Date/Price** anchor point coordinates are specified using the date and price values;
- **OBJPROP_CORNER** defines the chart corner relative to which the anchor point coordinates are specified. Can be one of the 4 values of the ENUM_BASE_CORNER enumeration;
- **OBJPROP_ANCHOR** defines the anchor point in object itself and can be one of the 9 values of the ENUM_ANCHOR_POINT enumeration. Coordinates in pixels are specified from this very point to selected chart corner;
- **OBJPROP_ANGLE** defines the object rotation angle counterclockwise.

The necessary variant can be specified using the function ObjectSetInteger(chart_handle, object_name, OBJPROP_ANCHOR, anchor_point_mode), where anchor_point_mode is one of the values of ENUM_ANCHOR_POINT.

ENUM_ANCHOR_POINT

ID	Description
ANCHOR_LEFT_UPPER	Anchor point at the upper left corner
ANCHOR_LEFT	Anchor point to the left in the center
ANCHOR_LEFT_LOWER	Anchor point at the lower left corner
ANCHOR_LOWER	Anchor point below in the center
ANCHOR_RIGHT_LOWER	Anchor point at the lower right corner
ANCHOR_RIGHT	Anchor point to the right in the center
ANCHOR_RIGHT_UPPER	Anchor point at the upper right corner
ANCHOR_UPPER	Anchor point above in the center
ANCHOR_CENTER	Anchor point strictly in the center of the object

The OBJ_BUTTON, OBJ_RECTANGLE_LABEL and OBJ_EDIT objects have a fixed anchor point in the upper left corner (ANCHOR_LEFT_UPPER).

Example:

```
void OnStart()   {
   string text_name="my_OBJ_TEXT_object";
   if(ObjectFind(0,text_name)<0)
     {
      Print("Object ",text_name," not found. Error code = ",GetLastError()
      //--- Get the maximal price of the chart
      double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
      //--- Create object Label
      ObjectCreate(0,text_name,OBJ_TEXT,0,TimeCurrent(),chart_max_price);
      //--- Set color of the text
      ObjectSetInteger(0,text_name,OBJPROP_COLOR,clrWhite);
      //--- Set background color
      ObjectSetInteger(0,text_name,OBJPROP_BGCOLOR,clrGreen);
      //--- Set text for the Label object
      ObjectSetString(0,text_name,OBJPROP_TEXT,TimeToString(TimeCurrent())
      //--- Set text font
      ObjectSetString(0,text_name,OBJPROP_FONT,"Trebuchet MS");
      //--- Set font size
      ObjectSetInteger(0,text_name,OBJPROP_FONTSIZE,10);
      //--- Bind to the upper right corner
      ObjectSetInteger(0,text_name,OBJPROP_ANCHOR,ANCHOR_RIGHT_UPPER);
      //--- Rotate 90 degrees counter-clockwise
      ObjectSetDouble(0,text_name,OBJPROP_ANGLE,90);
      //--- Forbid the selection of the object by mouse
      ObjectSetInteger(0,text_name,OBJPROP_SELECTABLE,false);
      //--- redraw object
      ChartRedraw(0);
     }
  }
```

Graphical objects Arrow (OBJ_ARROW) have only 2 ways of linking their coordinates. Identifiers are listed in ENUM_ARROW_ANCHOR.

ENUM_ARROW_ANCHOR

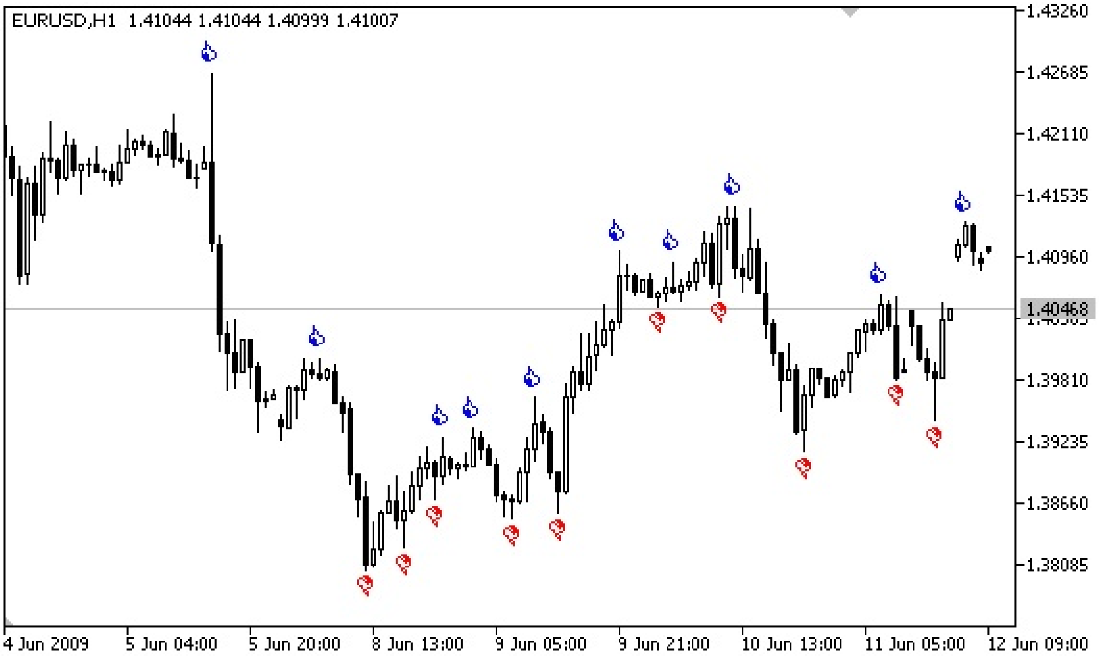
ID	Description
ANCHOR_TOP	Anchor on the top side
ANCHOR_BOTTOM	Anchor on the bottom side

Example:

```
#property strict
void OnStart()
  {
//--- Auxiliary arrays
   double Ups[],Downs[];
   datetime Times[];
```

```
//--- Set the arrays as timeseries
   ArraySetAsSeries(Ups,true);
   ArraySetAsSeries(Downs,true);
   ArraySetAsSeries(Times,true);
//--- Set Last error value to Zero
   ResetLastError();
//--- Copy timeseries containing the opening bars of the last 1000 ones
   int copied=CopyTime(NULL,0,0,1000,Times);
   if(copied<=0)
     {
      Print("Unable to copy the Open Time of the last 1000 bars");
      return;
     }
//--- prepare the Ups[] and Downs[] arrays
   ArrayResize(Ups,copied);
   ArrayResize(Downs,copied);
//--- copy the values of iFractals indicator
   for(int i=0;i<copied;i++)
   {
      Ups[i]=iFractals(NULL,0,MODE_UPPER,i);
    Downs[i]=iFractals(NULL,0,MODE_LOWER,i);
   }
//---
   int upcounter=0,downcounter=0; // count there the number of arrows
   bool created;// the result of attempts to create an object
   for(int i=2;i<copied;i++)// Run through the values of the indicator iFr
     {
      if(Ups[i]!=0)// Found the upper fractal
        {
         if(upcounter<10)// Create no more than 10 "Up" arrows
           {
            //--- Try to create an "Up" object
            created=ObjectCreate(0,string(Times[i]),OBJ_ARROW_THUMB_UP,0,T
            if(created)// If set up - let's make tuning for it
              {
               //--- Point anchor is below in order not to cover bar
               ObjectSetInteger(0,string(Times[i]),OBJPROP_ANCHOR,ANCHOR_B
               //--- Final touch - painted
               ObjectSetInteger(0,string(Times[i]),OBJPROP_COLOR,clrBlue);
               upcounter++;
              }
           }
        }
      if(Downs[i]!=0)// Found a lower fractal
        {
         if(downcounter<10)// Create no more than 10 arrows "Down"
           {
            //--- Try to create an object "Down"
            created=ObjectCreate(0,string(Times[i]),OBJ_ARROW_THUMB_DOWN,0
            if(created)// If set up - let's make tuning for it
              {
               //--- Point anchor is above in order not to cover bar
               ObjectSetInteger(0,string(Times[i]),OBJPROP_ANCHOR,ANCHOR_T
               //--- Final touch - painted
               ObjectSetInteger(0,string(Times[i]),OBJPROP_COLOR,clrRed);
               downcounter++;
              }
           }
        }
     }
  }
```

After the script execution the chart will look like in this figure.



The Chart Corner to Which an Object Is Attached

There is a number of graphical objects for which you can set a chart corner, relative to which the coordinates are specified in pixels. These are the following types of objects (in brackets object type identifiers are specified):

- Label (OBJ_LABEL);
- Button (OBJ_BUTTON);
- Bitmap Label (OBJ_BITMAP_LABEL);
- Edit (OBJ_EDIT).
- Rectangle Label (OBJ_RECTANGLE_LABEL);

Object	ID	X/Y	Width/Height	Date/Price	OBJPROP_C
Text	OBJ_TEXT			Yes	
Label	OBJ_LABEL	Yes	Yes (read only)		**Yes**
Button	OBJ_BUTTON	Yes	Yes		**Yes**
Bitmap	OBJ_BITMAP		Yes (read only)	Yes	
Bitmap Label	OBJ_BITMAP_LABEL	Yes	Yes (read only)		**Yes**
Edit	OBJ_EDIT	Yes	Yes		**Yes**
Rectangle Label	OBJ_RECTANGLE_LABEL	Yes	Yes		**Yes**

The following designations are used in the table:

- **X/Y** coordinates of anchor points specified in pixels relative to a chart corner;
- **Width/Height** objects have width and height. For "read only", the width and height values are calculated only once the object is rendered on chart;
- **Date/Price** anchor point coordinates are specified using the date and price values;
- **OBJPROP_CORNER** defines the chart corner relative to which the anchor point coordinates are specified. Can be one of the 4 values of the ENUM_BASE_CORNER enumeration;
- **OBJPROP_ANCHOR** defines the anchor point in object itself and can be one of the 9 values of the ENUM_ANCHOR_POINT enumeration. Coordinates in pixels are specified from this very point to selected chart corner;

· **OBJPROP_ANGLE** defines the object rotation angle counterclockwise.

In order to specify the chart corner, from which X and Y coordinates will be measured in pixels, use ObjectSetInteger(chartID, name, OBJPROP_CORNER, **chart_corner**), where:

· chartID - chart identifier;

· name name of a graphical object;

· OBJPROP_CORNER property ID to specify the corner for binding;

· **chart_corner** the desired chart corner, can be one of the values of the ENUM_BASE_CORNER enumeration.

ENUM_BASE_CORNER

ID	Description
CORNER_LEFT_UPPER	Center of coordinates is in the upper left corner of the chart
CORNER_LEFT_LOWER	Center of coordinates is in the lower left corner of the chart
CORNER_RIGHT_LOWER	Center of coordinates is in the lower right corner of the chart
CORNER_RIGHT_UPPER	Center of coordinates is in the upper right corner of the chart

Example:

```
void CreateLabel(long   chart_id,                  string name,
                 int    chart_corner,
                 int    anchor_point,
                 string text_label,
                 int    x_ord,
                 int    y_ord)
  {
//---
   if(ObjectCreate(chart_id,name,OBJ_LABEL,0,0,0))
     {
      ObjectSetInteger(chart_id,name,OBJPROP_CORNER,chart_corner);
      ObjectSetInteger(chart_id,name,OBJPROP_ANCHOR,anchor_point);
      ObjectSetInteger(chart_id,name,OBJPROP_XDISTANCE,x_ord);
      ObjectSetInteger(chart_id,name,OBJPROP_YDISTANCE,y_ord);
      ObjectSetString(chart_id,name,OBJPROP_TEXT,text_label);
     }
   else
      Print("Failed to create the object OBJ_LABEL ",name,", Error code =
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int height=(int)ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
   int width=(int)ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
   string arrows[4]={"LEFT_UPPER","RIGHT_UPPER","RIGHT_LOWER","LEFT_LOWER"
   CreateLabel(0,arrows[0],CORNER_LEFT_UPPER,ANCHOR_LEFT_UPPER,arrows[0],5
   CreateLabel(0,arrows[1],CORNER_RIGHT_UPPER,ANCHOR_RIGHT_UPPER,arrows[1]
   CreateLabel(0,arrows[2],CORNER_RIGHT_LOWER,ANCHOR_RIGHT_LOWER,arrows[2]
   CreateLabel(0,arrows[3],CORNER_LEFT_LOWER,ANCHOR_LEFT_LOWER,arrows[3],5
  }
```

Visibility of Objects

The combination of object visibility flags determines chart timeframes, where the object is visible. To set/get the value of the OBJPROP_TIMEFRAMES property, you can use the ObjectSet()/ObjectGet or ObjectSetInteger()/ObjectGetInteger() functions.

ID	Value	Description
OBJ_NO_PERIODS, EMPTY	-1	The object is not drawn in all timeframes
OBJ_PERIOD_M1	0x0001	The object is drawn in 1-minute chart
OBJ_PERIOD_M5	0x0002	The object is drawn in 5-minute chart
OBJ_PERIOD_M15	0x0004	The object is drawn in 15-minute chart
OBJ_PERIOD_M30	0x0008	The object is drawn in 30-minute chart
OBJ_PERIOD_H1	0x0010	The object is drawn in 1-hour chart
OBJ_PERIOD_H4	0x0020	The object is drawn in 4-hour chart
OBJ_PERIOD_D1	0x0040	The object is drawn in day charts
OBJ_PERIOD_W1	0x0080	The object is drawn in week charts
OBJ_PERIOD_MN1	0x0100	The object is drawn in month charts
OBJ_ALL_PERIODS	0x01ff	The object is drawn in all timeframes

Visibility flags can be combined using the symbol "|", for example, the combination of flags OBJ_PERIOD_M15|OBJ_PERIOD_H1 means that the object will be visible on the 15-minute and hourly timeframes.

Example:

```
void OnStart()   {
//---
   string highlevel="PreviousDayHigh";
   string lowlevel="PreviousDayLow";
   double prevHigh;           // The previous day High
   double prevLow;            // The previous day Low
   double highs[],lows[];     // Arrays for High and Low

//--- Reset the last error
   ResetLastError();
//--- Get the last 2 High values on the daily timeframe
   int highsgot=CopyHigh(Symbol(),PERIOD_D1,0,2,highs);
   if(highsgot>0) // If copying was successful
```

```
     {
      Print("High prices for the last 2 days were obtained successfully");
      prevHigh=highs[0]; // The previous day High
      Print("prevHigh = ",prevHigh);
      if(ObjectFind(0,highlevel)<0) // Object with the name highlevel not
        {
         ObjectCreate(0,highlevel,OBJ_HLINE,0,0,0); // Create the Horizont
        }
      //--- Set value for the price level for the line highlevel
      ObjectSetDouble(0,highlevel,OBJPROP_PRICE,0,prevHigh);
      //--- Set the visibility only PERIOD_M15 and PERIOD_H1
      ObjectSetInteger(0,highlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M15|OBJ_P
     }
   else
     {
      Print("Could not get High prices over the past 2 days, Error = ",Get
     }

//--- Reset the last error
   ResetLastError();
//--- Get the 2 days values Low on the daily timeframe
   int lowsgot=CopyLow(Symbol(),PERIOD_D1,0,2,lows);
   if(lowsgot>0) // If copying was successful
     {
      Print("Low prices for the last 2 days were obtained successfully");
      prevLow=lows[0]; // The previous day Low
      Print("prevLow = ",prevLow);
      if(ObjectFind(0,lowlevel)<0) // Object with the name lowlevel not fo
        {
         ObjectCreate(0,lowlevel,OBJ_HLINE,0,0,0); // Create the Horizonta
        }
      //--- Set value for the price level for the line lowlevel
      ObjectSetDouble(0,lowlevel,OBJPROP_PRICE,0,prevLow);
      //--- Set the visibility only PERIOD_M15 and PERIOD_H1
      ObjectSetInteger(0,lowlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M15|OBJ_PE
     }
   else Print("Could not get Low prices for the last 2 days, Error = ",Get

   ChartRedraw(0); // redraw the chart forcibly
  }
```

See also

PeriodSeconds, Period, Chart timeframes, Date and Time

Gann Objects

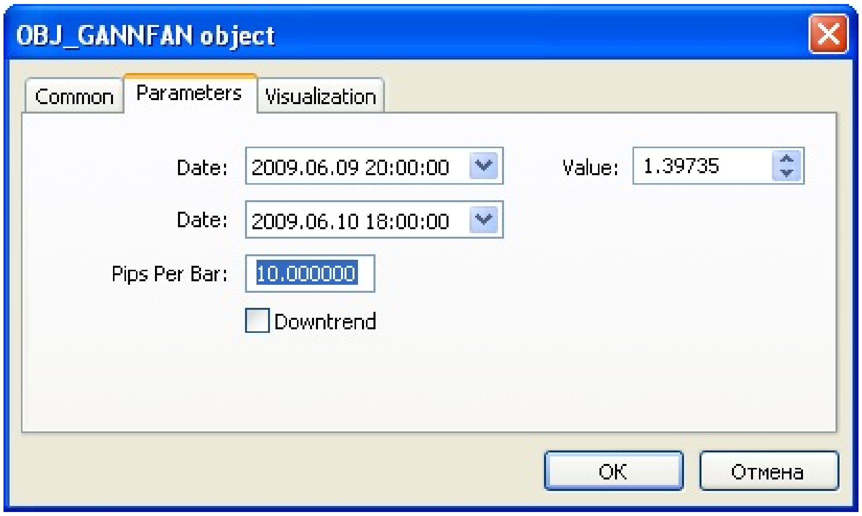
For Gann Fan (OBJ_GANNFAN) and Gann Grid (OBJ_GANNGRID) objects you can specify two values of the ENUM_GANN_DIRECTION enumeration that sets the trend direction.

ENUM_GANN_DIRECTION

ID	Description
GANN_UP_TREND	Line corresponding to the uptrend line
GANN_DOWN_TREND	Line corresponding to the downward trend

To set the scale of the main line as 1x1, use function ObjectSetDouble(chart_handle, gann_object_name, OBJPROP_SCALE, scale), where:

- chart_handle chart window where the object is located;
- gann_object_name object name;
- OBJPROP_SCALE identifier of the "Scale" property;
- scale required scale in units of Pips/Bar.



Date: 2009.06.09 20:00:00

Value: 1.39735

Date: 2009.06.10 18:00:00

Pips Per Bar: 10.000000

□ Downtrend

Example of creating Gann Fan:

```
void OnStart()   {
//---
   string my_gann="OBJ_GANNFAN object";
   if(ObjectFind(0,my_gann)<0)// Object not found
     {
      //--- Inform about the failure
      Print("Object ",my_gann," not found. Error code = ",GetLastError());
      //--- Get the maximal price of the chart
      double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
      //--- Get the minimal price of the chart
      double chart_min_price=ChartGetDouble(0,CHART_PRICE_MIN,0);
      //--- How many bars are shown in the chart?
      int bars_on_chart=int(ChartGetInteger(0,CHART_VISIBLE_BARS));
      //--- Create an array, to write the opening time of each bar to
      datetime Times[];
      //--- Arrange access to the array as that of timeseries
      ArraySetAsSeries(Times,true);
      //--- Now copy data of bars visible in the chart into this array
      int times=CopyTime(NULL,0,0,bars_on_chart,Times);
      if(times<=0)
        {
         Print("Could not copy the array with the open time!");
         return;
        }
      //--- Preliminary preparations completed

      //--- Index of the central bar in the chart
      int center_bar=bars_on_chart/2;
      //--- Chart equator - between the maximum and minimum
      double mean=(chart_max_price+chart_min_price)/2.0;
      //--- Set the coordinates of the first anchor point to the center
      ObjectCreate(0,my_gann,OBJ_GANNFAN,0,Times[center_bar],mean,
                   //--- Second anchor point to the right
                   Times[center_bar/2],(mean+chart_min_price)/2.0);
      Print("Times[center_bar] = "+(string)Times[center_bar]+"  Times[cent
      //Print("Times[center_bar]/="+Times[center_bar]+"  Times[center_bar/
      //--- Set the scale in units of Pips / Bar
      ObjectSetDouble(0,my_gann,OBJPROP_SCALE,10);
      //--- Set the line trend
      ObjectSetInteger(0,my_gann,OBJPROP_DIRECTION,GANN_UP_TREND);
      //--- Set the line width
      ObjectSetInteger(0,my_gann,OBJPROP_WIDTH,1);
      //--- Define the line style
      ObjectSetInteger(0,my_gann,OBJPROP_STYLE,STYLE_DASHDOT);
      //--- Set the line color
      ObjectSetInteger(0,my_gann,OBJPROP_COLOR,clrYellowGreen);
      //--- Allow the user to select an object
      ObjectSetInteger(0,my_gann,OBJPROP_SELECTABLE,true);
      //--- Select it yourself
      ObjectSetInteger(0,my_gann,OBJPROP_SELECTED,true);
      //--- Draw it on the chart
      ChartRedraw(0);
     }
  }
```

Web Colors

The following color constants are defined for the color type:

clrBlack	clrDarkGreen	clrDarkSlateGray	clrOlive	clrG
clrMaroon	clrIndigo	clrMidnightBlue	clrDarkBlue	clrDarkO
clrSeaGreen	clrDarkGoldenrod	clrDarkSlateBlue	clrSienna	clrMed
clrLightSeaGreen	clrDarkViolet	clrFireBrick	clrMediumVioletRed	clrMediu
clrGoldenrod	clrMediumSpringGreen	clrLawnGreen	clrCadetBlue	clrDark
clrDarkOrange	clrOrange	clrGold	clrYellow	clrCha
clrDeepSkyBlue	clrBlue	clrMagenta	clrRed	clrG
clrLightSlateGray	clrDeepPink	clrMediumTurquoise	clrDodgerBlue	clrTur
clrIndianRed	clrMediumOrchid	clrGreenYellow	clrMediumAquamarine	clrDarkS
clrMediumPurple	clrPaleVioletRed	clrCoral	clrCornflowerBlue	clrDar
clrDarkSalmon	clrBurlyWood	clrHotPink	clrSalmon	clrV
clrPlum	clrKhaki	clrLightGreen	clrAquamarine	clrS
clrPaleGreen	clrThistle	clrPowderBlue	clrPaleGoldenrod	clrPaleT
clrMoccasin	clrLightPink	clrGainsboro	clrPeachPuff	clrP
clrLemonChiffon	clrBeige	clrAntiqueWhite	clrPapayaWhip	clrCo
clrLavender	clrMistyRose	clrOldLace	clrWhiteSmoke	clrSe
clrLavenderBlush	clrMintCream	clrSnow	clrWhite	

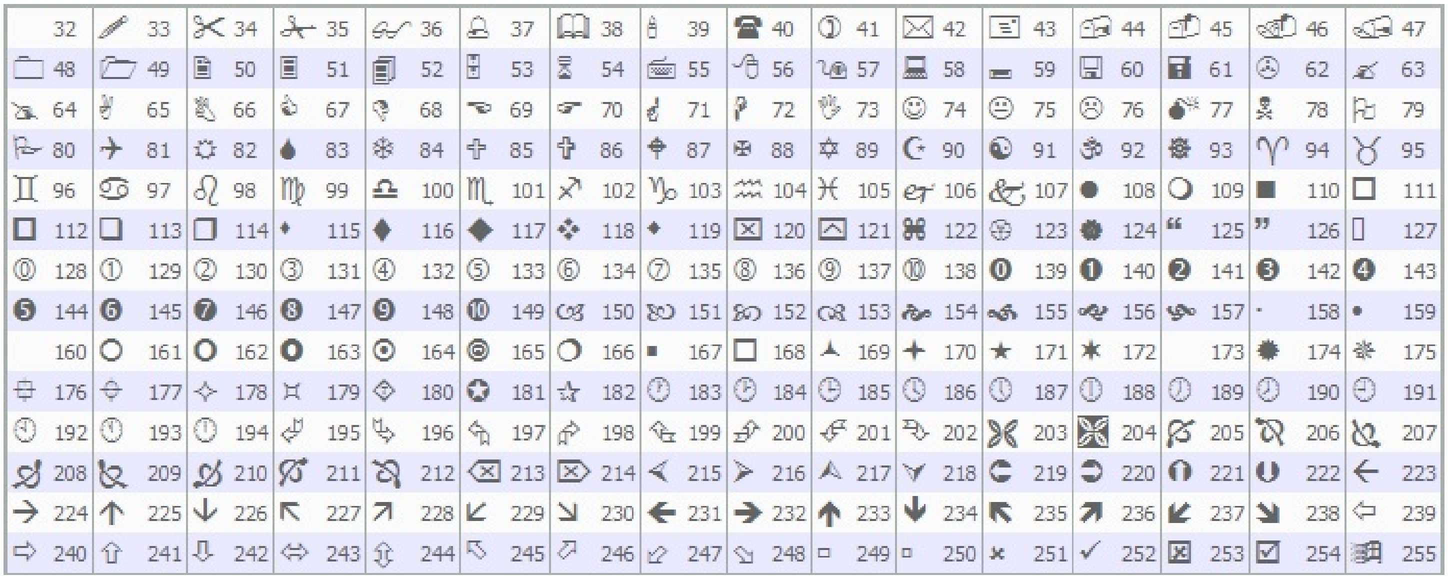
Color can be set to an object using the ObjectSetInteger() function. For getting color values there are similar functions ObjectGetInteger().

Example:

```
//---- indicator settings #property indicator_chart_window
#property indicator_buffers 3
#property indicator_type1   DRAW_LINE
#property indicator_type2   DRAW_LINE
#property indicator_type3   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_color2  clrRed
#property indicator_color3  clrLime
```

Wingdings

Characters of Wingdings used with the OBJ_ARROW object:



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	⑤ 133	⑥ 134	⑦ 135	⑧ 136	⑨ 137	⑩ 138	⓿ 139	❶ 140	❷ 141	❸ 142	❹ 143
❺ 144	❻ 145	❼ 146	❽ 147	❾ 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

A necessary character can be set using the ObjectSetInteger() function.

Example:

```
void OnStart()   {
//---
   string up_arrow="up_arrow";
   datetime time=TimeCurrent();
   double lastClose[1];
   int close=CopyClose(Symbol(),Period(),0,1,lastClose);     // Get the Cl
//--- If the price was obtained
   if(close>0)
     {
      ObjectCreate(0,up_arrow,OBJ_ARROW,0,0,0,0,0);          // Create an
      ObjectSetInteger(0,up_arrow,OBJPROP_ARROWCODE,241);    // Set the ar
      ObjectSetInteger(0,up_arrow,OBJPROP_TIME,time);        // Set time
      ObjectSetDouble(0,up_arrow,OBJPROP_PRICE,lastClose[0]);// Set price
      ChartRedraw(0);                                        // Draw arrow
     }
   else
      Print("Unable to get the latest Close price!");
  }
```

Arrow codes

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

ID	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:

ID	Value	Description
	1	Upwards arrow with tip rightwards
	2	Downwards arrow with tip rightwards
	3	Left pointing triangle
	4	Dash symbol
SYMBOL_LEFTPRICE	5	Left sided price label
SYMBOL_RIGHTPRICE	6	Right sided price label

Special Arrow codes cannot be used in custom indicators for lines with DRAW_ARROW drawing style.

Example:

```
#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
   SetIndexStyle(0,DRAW_ARROW);
   SetIndexArrow(0,159);
   SetIndexBuffer(0,SarBuffer);
//----
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| SaveLastReverse                                                  |
//+------------------------------------------------------------------+
void SaveLastReverse(int last,int dir,double start_index,double low,double
  {
   save_lastreverse=last;
   save_dirlong=dir;
   save_start=start_index;
   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_index,last_high,last_low;
   double ep,sar,price_low,price_high,price;
   int    i;
   int limit,cb;
//----
   if(Bars<3) return(0);
   int counted_bars=IndicatorCounted();
   if(counted_bars < 0)  return(-1);
   if(counted_bars>0) counted_bars--;
   limit=Bars-counted_bars;
   if(counted_bars==0) limit-=2;
```

```
//---- initial settings
   i=limit;
   if(counted_bars==0 || first)
     {
      first=false;
      dirlong=true;
      start_index=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; b
         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;
      start_index=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_index,price_low,last_high,ep,sar);
         start_index=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_index,last_low,price_high,ep,sar);
         start_index=Step; dirlong=true;
         ep=price_high; last_high=price_high;
         SarBuffer[i]=last_low;
         i--;
         continue;
        }
      //---
      price=SarBuffer[i+1];
      sar=price+start_index*(ep-price);
      if(dirlong)
        {
         if(ep<price_high && (start_index+Step)<=Maximum) start_index+=Ste
         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_index,price_low,last_high,ep,sar)
            start_index=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_index+Step)<=Maximum) start_index+=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_index,last_low,price_high,ep,sar
            start_index=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--;
     }
//---
   for(cb=limit;cb>=0;cb--)
     {
      if(GreaterDoubles(SarBuffer[cb],0,Digits))
        {
         string object_name="price"+string(Time[cb]);
         //--- first find object by name
         if(ObjectFind(object_name)<0)
           {
            //--- if not found, create it
            if(ObjectCreate(object_name,OBJ_ARROW,0,Time[cb],SarBuffer[cb]
              {
               //--- set object properties
               //--- arrow code
               ObjectSet(object_name,OBJPROP_ARROWCODE,SYMBOL_LEFTPRICE);
               //--- color
               ObjectSet(object_name,OBJPROP_COLOR,DodgerBlue);
               //--- price
               ObjectSet(object_name,OBJPROP_PRICE1,SarBuffer[cb]);
               //--- time
               ObjectSet(object_name,OBJPROP_TIME1,Time[cb]);
              }
           }
         else
           {
            //--- if the object exists, just modify its price coordinate
            ObjectSet(object_name,OBJPROP_PRICE1,SarBuffer[cb]);
           }
        }
     }
//----
   return(0);
  }
//+------------------------------------------------------------------+
//| GreaterDoubles                                                   |
//+------------------------------------------------------------------+
bool GreaterDoubles(double number1,double number2,int dig)
  {
   if(NormalizeDouble(number1-number2,dig)>0) return(true);
   else return(false);
  }
//+------------------------------------------------------------------+
```

Indicators Constants

There are many predefined technical indicators, which can be used in programs written in the MQL4 language. In addition, there is an opportunity to create custom indicators using the iCustom() function. All constants required for that are divided into 5 groups:

- Price constants for selecting the type of price or volume, on which an indicator is calculated;
- Series Array Identifiers - for selecting a value type from a timeseries;
- Smoothing methods built-in smoothing methods used in indicators;
- Indicator lines identifiers of indicator buffers when accessing indicator values;
- Drawing styles for indicating one of 6 types of drawing and setting the line drawing style;
- Custom indicators properties are used in functions for working with custom indicators;

Price Constants

Calculations of technical indicators require price values and/or values of volumes, on which calculations will be performed. There are 7 predefined identifiers from the ENUM_APPLIED_PRICE enumeration, used to specify the desired price base for calculations.

ENUM_APPLIED_PRICE

ID	Value	Description
PRICE_CLOSE	0	Close price
PRICE_OPEN	1	Open price
PRICE_HIGH	2	The maximum price for the period
PRICE_LOW	3	The minimum price for the period
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

The iStochastic() technical Indicator can be calculated in two ways using:

· either only Close prices;

· or High and Low prices.

To select a necessary variant for calculation, specify one of the values of the ENUM_STO_PRICE enumeration.

ENUM_STO_PRICE

ID	Description
STO_LOWHIGH	Calculation is based on Low/High prices
STO_CLOSECLOSE	Calculation is based on Close/Close prices

Series Array Identifiers

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

ID	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 iLowest() and iHighest() functions
MODE_TIME	5	Bar open time, used in ArrayCopySeries() function

Smoothing Methods

Many technical indicators are based on various methods of the price series smoothing. Some standard technical indicators (iAlligator(), iEnvelopes(), iEnvelopesOnArray(), iForce(), iGator(), iMA(), iMAOnArray(), iStdDev(), iStdDevOnArray() and iStochastic() indicators) require specification of the smoothing type as an input parameter. For specifying the desired type of smoothing, identifiers listed in the ENUM_MA_METHOD enumeration are used.

ENUM_MA_METHOD

ID	Value	Description
MODE_SMA	0	Simple averaging
MODE_EMA	1	Exponential averaging
MODE_SMMA	2	Smoothed averaging
MODE_LWMA	3	Linear-weighted averaging

Indicator Lines

Indicator line identifiers used in iMACD(), iRVI() and iStochastic() indicators.

ID	Value	Description
MODE_MAIN	0	Base indicator line
MODE_SIGNAL	1	Signal line

Indicator line identifiers used in iADX() indicator.

ID	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() and iFractals() indicators.

ID	Value	Description
MODE_UPPER	1	Upper line
MODE_LOWER	2	Lower line

Indicator line identifiers used in iAlligator() and iGator() indicators.

ID	Value	Description
MODE_GATORJAW	1	Jaw line
MODE_GATORTEETH	2	Teeth line
MODE_GATORLIPS	3	Lips line

Ichimoku Kinko Hyo identifiers used in iIchimoku() indicator call as source of requested data.

ID	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_CHIKOUSPAN	5	Chikou Span

Drawing Styles

Drawing shape styles for SetIndexStyle() function.

ID	Value	Description
DRAW_LINE	0	Drawing line
DRAW_SECTION	1	Drawing sections
DRAW_HISTOGRAM	2	Drawing histogram
DRAW_ARROW	3	Drawing arrows (symbols)
DRAW_ZIGZAG	4	Drawing sections between even and odd indicator buffers, 2 buffers of values
DRAW_NONE	12	No drawing

Drawing style enumeration for SetLevelStyle() function. Valid when width=1. It can be any of the following values:

ENUM_LINE_STYLE

ID	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

Drawing style also can be used for get/set the OBJPROP_STYLE property of the object.

When creating a custom indicator, you can specify one of 6 types of drawing styles (as displayed in the main chart window or a chart subwindow), whose values are specified above.

In one custom indicator, it is permissible to use any indicator drawing types. Each construction type requires specification of one to two global arrays for storing data necessary for drawing. These data arrays must be bound with indicator buffers using the SetIndexBuffer() function.

Custom Indicators Properties

A custom indicator has a lot of settings to provide convenient displaying. These settings are made through the assignment of corresponding indicator properties using functions IndicatorSetDouble(), IndicatorSetInteger() and IndicatorSetString(). Identifiers of indicator properties are listed in the ENUM_CUSTOMIND_PROPERTY enumeration.

ENUM_CUSTOMIND_PROPERTY_INTEGER

ID	Description	Property type
INDICATOR_DIGITS	Accuracy of drawing of indicator values	int
INDICATOR_HEIGHT	Fixed height of the indicator's window (the preprocessor command #property indicator_height)	int
INDICATOR_LEVELS	Number of levels in the indicator window	int
INDICATOR_LEVELCOLOR	Color of the level line	color sets color for all levels
INDICATOR_LEVELSTYLE	Style of the level line	ENUM_LINE_STYLE sets style for all levels
INDICATOR_LEVELWIDTH	Thickness of the level line	int sets width for all levels

ENUM_CUSTOMIND_PROPERTY_DOUBLE

ID	Description	Property type
INDICATOR_MINIMUM	Minimum of the indicator window	double
INDICATOR_MAXIMUM	Maximum of the indicator window	double
INDICATOR_LEVELVALUE	Level value	double modifier = level number

ENUM_CUSTOMIND_PROPERTY_STRING

ID	Description	Property type

INDICATOR_SHORTNAME	Short indicator name	string
INDICATOR_LEVELTEXT	Level description	string modifier = level number

Example:

```
#property description "Relative Strength Index" #property strict
#property indicator_separate_window
#property indicator_buffers 3
#property indicator_color1     clrDodgerBlue
//--- input parameters
input int InpRSIPeriod=14; // RSI Period
//--- buffers
double ExtRSIBuffer[];
double ExtPosBuffer[];
double ExtNegBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit(void)
  {
   string short_name;
//--- 2 additional buffers are used for calculations.
   IndicatorBuffers(3);
   SetIndexBuffer(0,ExtRSIBuffer);
   SetIndexBuffer(1,ExtPosBuffer);
   SetIndexBuffer(2,ExtNegBuffer);
//--- set levels
   IndicatorSetInteger(INDICATOR_LEVELS,2);
   IndicatorSetDouble(INDICATOR_LEVELVALUE,0,30);
   IndicatorSetDouble(INDICATOR_LEVELVALUE,1,70);
//--- set maximum and minimum for subwindow
   IndicatorSetDouble(INDICATOR_MINIMUM,0);
   IndicatorSetDouble(INDICATOR_MAXIMUM,100);
//--- indicator line
   SetIndexStyle(0,DRAW_LINE);
   SetIndexBuffer(0,ExtRSIBuffer);
//--- name for DataWindow and indicator subwindow label
   short_name="RSI("+string(InpRSIPeriod)+")";
   IndicatorShortName(short_name);
   SetIndexLabel(0,short_name);
//--- check for input
   if(InpRSIPeriod<2)
     {
      Print("Incorrect value for input variable InpRSIPeriod = ",InpRSIPer
      return(INIT_FAILED);
     }
```

```
//---
   SetIndexDrawBegin(0,InpRSIPeriod);
//--- initialization done
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Relative Strength Index                                          |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   int    i,pos;
   double diff;
//---
   if(Bars<=InpRSIPeriod || InpRSIPeriod<2)
      return(0);
//--- counting from 0 to rates_total
   ArraySetAsSeries(ExtRSIBuffer,false);
   ArraySetAsSeries(ExtPosBuffer,false);
   ArraySetAsSeries(ExtNegBuffer,false);
   ArraySetAsSeries(close,false);
//--- preliminary calculations
   pos=prev_calculated-1;
   if(pos<=InpRSIPeriod)
     {
      //--- first RSIPeriod values of the indicator are not calculated
      ExtRSIBuffer[0]=0.0;
      ExtPosBuffer[0]=0.0;
      ExtNegBuffer[0]=0.0;
      double sump=0.0;
      double sumn=0.0;
      for(i=1; i<=InpRSIPeriod; i++)
        {
         ExtRSIBuffer[i]=0.0;
         ExtPosBuffer[i]=0.0;
         ExtNegBuffer[i]=0.0;
         diff=close[i]-close[i-1];
         if(diff>0)
            sump+=diff;
         else
            sumn-=diff;
        }
      //--- calculate first visible value
      ExtPosBuffer[InpRSIPeriod]=sump/InpRSIPeriod;
      ExtNegBuffer[InpRSIPeriod]=sumn/InpRSIPeriod;
      if(ExtNegBuffer[InpRSIPeriod]!=0.0)
         ExtRSIBuffer[InpRSIPeriod]=100.0-(100.0/(1.0+ExtPosBuffer[InpRSIP
      else
        {
         if(ExtPosBuffer[InpRSIPeriod]!=0.0)
            ExtRSIBuffer[InpRSIPeriod]=100.0;
         else
            ExtRSIBuffer[InpRSIPeriod]=50.0;
        }
      //--- prepare the position value for main calculation
      pos=InpRSIPeriod+1;
     }
//--- the main loop of calculations
   for(i=pos; i<rates_total && !IsStopped(); i++)
     {
      diff=close[i]-close[i-1];
      ExtPosBuffer[i]=(ExtPosBuffer[i-1]*(InpRSIPeriod-1)+(diff>0.0?diff:0
      ExtNegBuffer[i]=(ExtNegBuffer[i-1]*(InpRSIPeriod-1)+(diff<0.0?-diff:
      if(ExtNegBuffer[i]!=0.0)
         ExtRSIBuffer[i]=100.0-100.0/(1+ExtPosBuffer[i]/ExtNegBuffer[i]);
      else
        {
         if(ExtPosBuffer[i]!=0.0)
            ExtRSIBuffer[i]=100.0;
         else
            ExtRSIBuffer[i]=50.0;
        }
     }
//---
   return(rates_total);
  }
```

Environment State

Constants describing the current runtime environment of an mql4-program are divided into groups:

· Client terminal properties information about the client terminal;

· Executed MQL5-program properties mql4 program properties, which help to control its execution;

· Symbol properties obtaining information about a symbol;

· Account properties information about the current account;

· Testing Statistics results of Expert Advisor testing.

Client Terminal Properties

Information about the client terminal can be obtained by two functions: TerminalInfoInteger() and TerminalInfoString(). For parameters, these functions accept values from ENUM_TERMINAL_INFO_INTEGER and ENUM_TERMINAL_INFO_STRING respectively.

ENUM_TERMINAL_INFO_INTEGER

Identifier	Description	Type
TERMINAL_BUILD	The client terminal build number	int
TERMINAL_COMMUNITY_ACCOUNT	The flag indicates the presence of MQL5.community authorization data in the terminal	bool
TERMINAL_COMMUNITY_CONNECTION	Connection to MQL5.community	bool
TERMINAL_CONNECTED	Connection to a trade server	bool
TERMINAL_DLLS_ALLOWED	Permission to use DLL	bool
TERMINAL_TRADE_ALLOWED	Permission to trade	bool
TERMINAL_EMAIL_ENABLED	Permission to send e-mails using SMTP-server and login, specified in the terminal settings	bool
TERMINAL_FTP_ENABLED	Permission to send reports using FTP-server and login, specified in the terminal settings	bool
TERMINAL_NOTIFICATIONS_ENABLED	Permission to send notifications to smartphone	bool
TERMINAL_MAXBARS	The maximal bars count on the chart	int
TERMINAL_MQID	The flag indicates the presence of MetaQuotes ID data to send Push notifications	bool
TERMINAL_CODEPAGE	Number of the code page of the language installed in the client terminal	int
TERMINAL_CPU_CORES	The number of CPU cores in the system	int
TERMINAL_DISK_SPACE	Free disk space for the MQL4\Files	int

	folder of the terminal, Mb	
TERMINAL_MEMORY_PHYSICAL	Physical memory in the system, Mb	int
TERMINAL_MEMORY_TOTAL	Memory available to the process of the terminal , Mb	int
TERMINAL_MEMORY_AVAILABLE	Free memory of the terminal process, Mb	int
TERMINAL_MEMORY_USED	Memory used by the terminal , Mb	int
TERMINAL_SCREEN_DPI	The resolution of information display on the screen is measured as number of Dots in a line per Inch (DPI). Knowing the parameter value, you can set the size of graphical objects so that they look the same on monitors with different resolution characteristics.	int
TERMINAL_PING_LAST	The last known value of a ping to a trade server in microseconds. One second comprises of one million microseconds	int

Example of scaling factor calculation:

```
//--- Creating a 1.5 inch wide button on a screen int screen_dpi = Termina
int base_width = 144;                                     // The basic wi
int width      = (button_width * screen_dpi) / 96;        // Calculate th
...

//--- Calculating the scaling factor as a percentage
int scale_factor=(TerminalInfoInteger(TERMINAL_SCREEN_DPI) * 100) / 96;
//--- Use of the scaling factor
width=(base_width * scale_factor) / 100;
```

In the above example, the graphical resource looks the same on monitors with different resolution characteristics. The size of control elements (buttons, dialog windows, etc.) corresponds to personalization settings.

ENUM_TERMINAL_INFO_DOUBLE

Identifier	Description	Type
TERMINAL_COMMUNITY_BALANCE	Balance in MQL5.community	double

File operations can be performed only in two directories; corresponding paths can be obtained using the request for TERMINAL_DATA_PATH and TERMINAL_COMMONDATA_PATH properties.

ENUM_TERMINAL_INFO_STRING

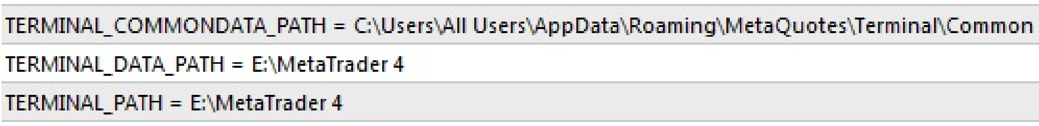
Identifier	Description	Type
TERMINAL_LANGUAGE	Language of the terminal	string
TERMINAL_COMPANY	Company name	string
TERMINAL_NAME	Terminal name	string
TERMINAL_PATH	Folder from which the terminal is started	string
TERMINAL_DATA_PATH	Folder in which terminal data are stored	string
TERMINAL_COMMONDATA_PATH	Common path for all of the terminals installed on a computer	string

For a better understanding of paths, stored in properties of TERMINAL_PATH, TERMINAL_DATA_PATH and TERMINAL_COMMONDATA_PATH parameters, it is recommended to execute the script, which will return these values for the current copy of the client terminal, installed on your computer.

Example: Script returns information about the client terminal paths

```
//+------------------------------------------------------------------+
//|                                          Check_TerminalPaths.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   Print("TERMINAL_PATH = ",TerminalInfoString(TERMINAL_PATH));
   Print("TERMINAL_DATA_PATH = ",TerminalInfoString(TERMINAL_DATA_PATH));
   Print("TERMINAL_COMMONDATA_PATH = ",TerminalInfoString(TERMINAL_COMMOND
  }
```

As result of the script execution in the Experts Journal you will see a messages, like the following:



```
TERMINAL_COMMONDATA_PATH = C:\Users\All Users\AppData\Roaming\MetaQuotes\Terminal\Common
TERMINAL_DATA_PATH = E:\MetaTrader 4
TERMINAL_PATH = E:\MetaTrader 4
```

Running MQL4 Program Properties

To obtain information about the currently running mql4 program, constants from ENUM_MQL_INFO_INTEGER and ENUM_MQL_INFO_STRING are used.

For function MQLInfoInteger

ENUM_MQL_INFO_INTEGER

Identifier	Description	Type
MQL_CODEPAGE	Codepage used by an MQL4 program to output and convert strings (Print, PrintFormat, Alert, MessageBox, SendFTP, SendMail, SendNotification, etc.)	Codepage constant
MQL_PROGRAM_TYPE	Type of the MQL4 program	ENUM_PROGRAM_TYPE
MQL_DLLS_ALLOWED	The permission to use DLL **for the given executed program**	bool
MQL_TRADE_ALLOWED	The permission to trade **for the given executed program**	bool
MQL_SIGNALS_ALLOWED	The permission to modify the Signals **for the given executed program**	bool
MQL_DEBUG	The flag, that indicates the debug mode	bool
MQL_PROFILER	The flag, that indicates the program operating in the code profiling mode	bool
MQL_TESTER	The flag, that indicates the tester process	bool
MQL_OPTIMIZATION	The flag, that indicates the optimization process	bool
MQL_VISUAL_MODE	The flag, that indicates the visual tester process	bool
MQL_FRAME_MODE	The flag, that indicates the Expert Advisor operating in gathering optimization result frames mode	bool
MQL_LICENSE_TYPE	Type of license of the EX4 module. The license refers to the EX4 module, from which a request is	ENUM_LICENSE_TYPE

	made using MQLInfoInteger(MQL_LICENSE_TYPE).	

For function MQLInfoString

ENUM_MQL_INFO_STRING

Identifier	Description	Type
MQL_PROGRAM_NAME	Name of the MQL4-program executed	string
MQL_PROGRAM_PATH	Path **for the given executed program**	string

For information about the type of the running program, values of ENUM_PROGRAM_TYPE are used.

ENUM_PROGRAM_TYPE

Identifier	Description
PROGRAM_SCRIPT	Script
PROGRAM_EXPERT	Expert
PROGRAM_INDICATOR	Indicator

ENUM_LICENSE_TYPE

Identifier	Description
LICENSE_FREE	A free unlimited version
LICENSE_DEMO	A trial version of a paid product from the Market. It works only in the strategy tester
LICENSE_FULL	A purchased licensed version allows at least 5 activations. The number of activations is specified by seller. Seller may increase the allowed number of activations
LICENSE_TIME	A version with limited term license

Example:

```
   ENUM_PROGRAM_TYPE mql_program=(ENUM_PROGRAM_TYPE)MQLInfoInteger(MQL_PRO
     {
      case PROGRAM_SCRIPT:
        {
         Print(__FILE__+" is script");
         break;
        }
      case PROGRAM_EXPERT:
        {
         Print(__FILE__+" is Expert Advisor");
         break;
        }
      case PROGRAM_INDICATOR:
        {
         Print(__FILE__+" is custom indicator");
         break;
        }
      default:Print("MQL4 program type value is ",mql_program);
     }
```

Symbol Properties

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

ID	Value	Description
MODE_LOW	1	Low day price
MODE_HIGH	2	High day price
MODE_TIME	5	The last incoming tick time (last known server time)
MODE_BID	9	Last incoming bid price. For the current symbol, it is stored in the predefined variable Bid
MODE_ASK	10	Last incoming ask price. For the current symbol, it is stored in the predefined variable Ask
MODE_POINT	11	Point size in the quote currency. For the current symbol, it is stored in the predefined variable Point
MODE_DIGITS	12	Count of digits after decimal point in the symbol prices. For the current symbol, it is stored in the predefined variable Digits
MODE_SPREAD	13	Spread value in points
MODE_STOPLEVEL	14	Stop level in points A zero value of MODE_STOPLEVEL means either absence of any restrictions on the minimal distance for Stop Loss/Take Profit or the fact that a trade server utilizes some external mechanisms for dynamic level control, which cannot be translated in the client terminal. In the second case, GetLastError() can return error 130, because MODE_STOPLEVEL is actually "floating" here.
MODE_LOTSIZE	15	Lot size in the base currency
MODE_TICKVALUE	16	Tick value in the deposit currency
MODE_TICKSIZE	17	Tick size in points

MODE_SWAPLONG	18	Swap of the buy order
MODE_SWAPSHORT	19	Swap of the sell order
MODE_STARTING	20	Market starting date (usually used for futures)
MODE_EXPIRATION	21	Market expiration date (usually used for futures)
MODE_TRADEALLOWED	22	Trade is allowed for the symbol
MODE_MINLOT	23	Minimum permitted amount of a lot
MODE_LOTSTEP	24	Step for changing lots
MODE_MAXLOT	25	Maximum permitted amount of a lot
MODE_SWAPTYPE	26	Swap calculation method. 0 - in points; 1 - in the symbol base currency; 2 - by interest; 3 - in the margin currency
MODE_PROFITCALCMODE	27	Profit calculation mode. 0 - Forex; 1 - CFD; 2 - Futures
MODE_MARGINCALCMODE	28	Margin calculation mode. 0 - Forex; 1 - CFD; 2 - Futures; 3 - CFD for indices
MODE_MARGININIT	29	Initial margin requirements for 1 lot
MODE_MARGINMAINTENANCE	30	Margin to maintain open orders calculated for 1 lot
MODE_MARGINHEDGED	31	Hedged margin calculated for 1 lot
MODE_MARGINREQUIRED	32	Free margin required to open 1 lot for buying
MODE_FREEZELEVEL	33	Order freeze level in points. If the execution price lies within the range defined by the freeze level, the order cannot be modified, cancelled or closed

Example:

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
void OnStart()
  {
   Print("Symbol=",Symbol());
   Print("Low day price=",MarketInfo(Symbol(),MODE_LOW));
   Print("High day price=",MarketInfo(Symbol(),MODE_HIGH));
   Print("The last incoming tick time=",(MarketInfo(Symbol(),MODE_TIME)));
   Print("Last incoming bid price=",MarketInfo(Symbol(),MODE_BID));
   Print("Last incoming ask price=",MarketInfo(Symbol(),MODE_ASK));
   Print("Point size in the quote currency=",MarketInfo(Symbol(),MODE_POIN
   Print("Digits after decimal point=",MarketInfo(Symbol(),MODE_DIGITS));
   Print("Spread value in points=",MarketInfo(Symbol(),MODE_SPREAD));
   Print("Stop level in points=",MarketInfo(Symbol(),MODE_STOPLEVEL));
   Print("Lot size in the base currency=",MarketInfo(Symbol(),MODE_LOTSIZE
   Print("Tick value in the deposit currency=",MarketInfo(Symbol(),MODE_TI
   Print("Tick size in points=",MarketInfo(Symbol(),MODE_TICKSIZE));
   Print("Swap of the buy order=",MarketInfo(Symbol(),MODE_SWAPLONG));
   Print("Swap of the sell order=",MarketInfo(Symbol(),MODE_SWAPSHORT));
   Print("Market starting date (for futures)=",MarketInfo(Symbol(),MODE_ST
   Print("Market expiration date (for futures)=",MarketInfo(Symbol(),MODE_
   Print("Trade is allowed for the symbol=",MarketInfo(Symbol(),MODE_TRADE
   Print("Minimum permitted amount of a lot=",MarketInfo(Symbol(),MODE_MIN
   Print("Step for changing lots=",MarketInfo(Symbol(),MODE_LOTSTEP));
   Print("Maximum permitted amount of a lot=",MarketInfo(Symbol(),MODE_MAX
   Print("Swap calculation method=",MarketInfo(Symbol(),MODE_SWAPTYPE));
   Print("Profit calculation mode=",MarketInfo(Symbol(),MODE_PROFITCALCMOD
   Print("Margin calculation mode=",MarketInfo(Symbol(),MODE_MARGINCALCMOD
   Print("Initial margin requirements for 1 lot=",MarketInfo(Symbol(),MODE
   Print("Margin to maintain open orders calculated for 1 lot=",MarketInfo
   Print("Hedged margin calculated for 1 lot=",MarketInfo(Symbol(),MODE_MA
   Print("Free margin required to open 1 lot for buying=",MarketInfo(Symbo
   Print("Order freeze level in points=",MarketInfo(Symbol(),MODE_FREEZELE
  }
```

To obtain the current market information there are several functions: SymbolInfoInteger(), SymbolInfoDouble() and SymbolInfoString(). The first parameter is the symbol name, the values of the second function parameter can be one of the identifiers of ENUM_SYMBOL_INFO_INTEGER, ENUM_SYMBOL_INFO_DOUBLE and ENUM_SYMBOL_INFO_STRING.

For function SymbolInfoInteger():

ENUM_SYMBOL_INFO_INTEGER

Identifier	Description	Type
SYMBOL_SELECT	Symbol is selected	bool

	in Market Watch. Some symbols can be hidden in Market Watch, but still they are considered as selected.	
SYMBOL_VISIBLE	Symbol is visible in Market Watch. Some symbols (mostly, these are cross rates required for calculation of margin requirements or profits in deposit currency) are selected automatically, but generally are not visible in Market Watch. To be displayed such symbols have to be explicitly selected.	bool
SYMBOL_SESSION_DEALS	Not supported	long
SYMBOL_SESSION_BUY_ORDERS	Not supported	long
SYMBOL_SESSION_SELL_ORDERS	Not supported	long
SYMBOL_VOLUME	Not supported	long
SYMBOL_VOLUMEHIGH	Not supported	long
SYMBOL_VOLUMELOW	Not supported	long
SYMBOL_TIME	Time of the last quote	datetime
SYMBOL_DIGITS	Digits after a decimal point	int
SYMBOL_SPREAD_FLOAT	Indication of a floating spread	bool

SYMBOL_SPREAD	Spread value in points	int
SYMBOL_TRADE_CALC_MODE	Contract price calculation mode	int
SYMBOL_TRADE_MODE	Order execution type	ENUM_SYMBOL_TRADE_MODE
SYMBOL_START_TIME	Date of the symbol trade beginning (usually used for futures)	datetime
SYMBOL_EXPIRATION_TIME	Date of the symbol trade end (usually used for futures)	datetime
SYMBOL_TRADE_STOPS_LEVEL	Minimal indention in points from the current close price to place Stop orders	int
SYMBOL_TRADE_FREEZE_LEVEL	Distance to freeze trade operations in points	int
SYMBOL_TRADE_EXEMODE	Deal execution mode	ENUM_SYMBOL_TRADE_EXECUTION
SYMBOL_SWAP_MODE	Swap calculation model	int
SYMBOL_SWAP_ROLLOVER3DAYS	Day of week to charge 3 days swap rollover	ENUM_DAY_OF_WEEK
SYMBOL_EXPIRATION_MODE	Not supported	int
SYMBOL_FILLING_MODE	Not supported	int
SYMBOL_ORDER_MODE	Not supported	int

For function SymbolInfoDouble():

ENUM_SYMBOL_INFO_DOUBLE

Identifier	Description	Type
SYMBOL_BID	Bid - best sell offer	double
SYMBOL_BIDHIGH	Not supported	double

SYMBOL_BIDLOW	Not supported	double
SYMBOL_ASK	Ask - best buy offer	double
SYMBOL_ASKHIGH	Not supported	double
SYMBOL_ASKLOW	Not supported	double
SYMBOL_LAST	Not supported	double
SYMBOL_LASTHIGH	Not supported	double
SYMBOL_LASTLOW	Not supported	double
SYMBOL_POINT	Symbol point value	double
SYMBOL_TRADE_TICK_VALUE	Value of SYMBOL_TRADE_TICK_VALUE_PROFIT	double
SYMBOL_TRADE_TICK_VALUE_PROFIT	Not supported	double
SYMBOL_TRADE_TICK_VALUE_LOSS	Not supported	double
SYMBOL_TRADE_TICK_SIZE	Minimal price change	double
SYMBOL_TRADE_CONTRACT_SIZE	Trade contract size	double
SYMBOL_VOLUME_MIN	Minimal volume for a deal	double
SYMBOL_VOLUME_MAX	Maximal volume for a deal	double
SYMBOL_VOLUME_STEP	Minimal volume change step for deal execution	double
SYMBOL_VOLUME_LIMIT	Not supported	double
SYMBOL_SWAP_LONG	Buy order swap value	double
SYMBOL_SWAP_SHORT	Sell order swap value	double
SYMBOL_MARGIN_INITIAL	Initial margin means the amount in the margin currency required for opening an order with the volume of one lot. It is used for checking a client's assets when he or she enters the market.	double
SYMBOL_MARGIN_MAINTENANCE	The maintenance margin. If it is set, it sets the margin amount in the margin currency of the symbol, charged from one lot. It is used for checking a client's assets when his/her account state changes. If the maintenance margin is equal to 0, the initial margin is used.	double

SYMBOL_MARGIN_LONG	Not supported	double
SYMBOL_MARGIN_SHORT	Not supported	double
SYMBOL_MARGIN_LIMIT	Not supported	double
SYMBOL_MARGIN_STOP	Not supported	double
SYMBOL_MARGIN_STOPLIMIT	Not supported	double
SYMBOL_SESSION_VOLUME	Not supported	double
SYMBOL_SESSION_TURNOVER	Not supported	double
SYMBOL_SESSION_INTEREST	Not supported	double
SYMBOL_SESSION_BUY_ORDERS_VOLUME	Not supported	double
SYMBOL_SESSION_SELL_ORDERS_VOLUME	Not supported	double
SYMBOL_SESSION_OPEN	Not supported	double
SYMBOL_SESSION_CLOSE	Not supported	double
SYMBOL_SESSION_AW	Not supported	double
SYMBOL_SESSION_PRICE_SETTLEMENT	Not supported	double
SYMBOL_SESSION_PRICE_LIMIT_MIN	Not supported	double
SYMBOL_SESSION_PRICE_LIMIT_MAX	Not supported	double

For function SymbolInfoString():

ENUM_SYMBOL_INFO_STRING

Identifier	Description	Type
SYMBOL_CURRENCY_BASE	Basic currency of a symbol	string
SYMBOL_CURRENCY_PROFIT	Profit currency	string
SYMBOL_CURRENCY_MARGIN	Margin currency	string
SYMBOL_DESCRIPTION	Symbol description	string
SYMBOL_PATH	Path in the symbol tree	string

There are several symbol trading modes. Information about trading modes of a certain symbol is reflected in the values of enumeration ENUM_SYMBOL_TRADE_MODE.

ENUM_SYMBOL_TRADE_MODE

Identifier	Description
SYMBOL_TRADE_MODE_DISABLED	Trade is disabled for the symbol

SYMBOL_TRADE_MODE_LONGONLY*	Allowed only long positions
SYMBOL_TRADE_MODE_SHORTONLY*	Allowed only short positions
SYMBOL_TRADE_MODE_CLOSEONLY	Allowed only position close operations
SYMBOL_TRADE_MODE_FULL	No trade restrictions

Possible deal execution modes for a certain symbol are defined in enumeration ENUM_SYMBOL_TRADE_EXECUTION.

ENUM_SYMBOL_TRADE_EXECUTION

Identifier	Description
SYMBOL_TRADE_EXECUTION_REQUEST	Execution by request
SYMBOL_TRADE_EXECUTION_INSTANT	Instant execution
SYMBOL_TRADE_EXECUTION_MARKET	Market execution
SYMBOL_TRADE_EXECUTION_EXCHANGE*	Exchange execution

*These values are not used in MQL4 (added for compatibility with MQL5).

Values of the ENUM_DAY_OF_WEEK enumeration are used for specifying days of week.

ENUM_DAY_OF_WEEK

Identifier	Description
SUNDAY	Sunday
MONDAY	Monday
TUESDAY	Tuesday
WEDNESDAY	Wednesday
THURSDAY	Thursday
FRIDAY	Friday
SATURDAY	Saturday

Account Properties

To obtain information about the current account there are several functions: AccountInfoInteger(), AccountInfoDouble() and AccountInfoString(). The function parameter values can accept values from the corresponding ENUM_ACCOUNT_INFO enumerations.

For the function AccountInfoInteger()

ENUM_ACCOUNT_INFO_INTEGER

Identifier	Description	Type
ACCOUNT_LOGIN	Account number	long
ACCOUNT_TRADE_MODE	Account trade mode	ENUM_ACCOUNT_TRADE_MODE
ACCOUNT_LEVERAGE	Account leverage	long
ACCOUNT_LIMIT_ORDERS	Maximum allowed number of active pending orders (0-unlimited)	int
ACCOUNT_MARGIN_SO_MODE	Mode for setting the minimal allowed margin	ENUM_ACCOUNT_STOPOUT_MODE
ACCOUNT_TRADE_ALLOWED	Allowed trade for the current account	bool
ACCOUNT_TRADE_EXPERT	Allowed trade for an Expert Advisor	bool

For the function AccountInfoDouble()

ENUM_ACCOUNT_INFO_DOUBLE

Identifier	Description	Type
ACCOUNT_BALANCE	Account balance in the deposit currency	double
ACCOUNT_CREDIT	Account credit in the deposit currency	double
ACCOUNT_PROFIT	Current profit of an account in the deposit currency	double
ACCOUNT_EQUITY	Account equity in the deposit currency	double

ACCOUNT_MARGIN	Account margin used in the deposit currency	double
ACCOUNT_MARGIN_FREE	Free margin of an account in the deposit currency	double
ACCOUNT_MARGIN_LEVEL	Account margin level in percents	double
ACCOUNT_MARGIN_SO_CALL	Margin call level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	double
ACCOUNT_MARGIN_SO_SO	Margin stop out level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	double
ACCOUNT_MARGIN_INITIAL	Not supported	double
ACCOUNT_MARGIN_MAINTENANCE	Not supported	double
ACCOUNT_ASSETS	Not supported	double
ACCOUNT_LIABILITIES	Not supported	double
ACCOUNT_COMMISSION_BLOCKED	Not supported	double

For function AccountInfoString()

ENUM_ACCOUNT_INFO_STRING

Identifier	Description	Type
ACCOUNT_NAME	Client name	string
ACCOUNT_SERVER	Trade server name	string
ACCOUNT_CURRENCY	Account currency	string
ACCOUNT_COMPANY	Name of a company that serves the account	string

There are several types of accounts that can be opened on a trade server. The type of account on which an MQL4 program is running can be found out using the ENUM_ACCOUNT_TRADE_MODE enumeration.

ENUM_ACCOUNT_TRADE_MODE

Identifier	Description

ACCOUNT_TRADE_MODE_DEMO	Demo account
ACCOUNT_TRADE_MODE_CONTEST	Contest account
ACCOUNT_TRADE_MODE_REAL	Real account

In case equity is not enough for maintaining open orders, the Stop Out situation, i.e. forced closing occurs. The minimum margin level at which Stop Out occurs can be set in percentage or in monetary terms. To find out the mode set for the account use the ENUM_ACCOUNT_STOPOUT_MODE enumeration.

ENUM_ACCOUNT_STOPOUT_MODE

Identifier	Description
ACCOUNT_STOPOUT_MODE_PERCENT	Account stop out mode in percents
ACCOUNT_STOPOUT_MODE_MONEY	Account stop out mode in money

An example of the script that outputs a brief account information.

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Name of the company
   string company=AccountInfoString(ACCOUNT_COMPANY);
//--- Name of the client
   string name=AccountInfoString(ACCOUNT_NAME);
//--- Account number
   long login=AccountInfoInteger(ACCOUNT_LOGIN);
//--- Name of the server
   string server=AccountInfoString(ACCOUNT_SERVER);
//--- Account currency
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- Demo, contest or real account
   ENUM_ACCOUNT_TRADE_MODE account_type=(ENUM_ACCOUNT_TRADE_MODE)AccountIn
//--- Now transform the value of  the enumeration into an understandable f
   string trade_mode;
   switch(account_type)
     {
      case  ACCOUNT_TRADE_MODE_DEMO:
         trade_mode="demo";
         break;
      case  ACCOUNT_TRADE_MODE_CONTEST:
         trade_mode="contest";
         break;
      default:
         trade_mode="real";
         break;
     }
//--- Stop Out is set in percentage or money
   ENUM_ACCOUNT_STOPOUT_MODE stop_out_mode=(ENUM_ACCOUNT_STOPOUT_MODE)Accou
//--- Get the value of the levels when Margin Call and Stop Out occur
   double margin_call=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL);
   double stop_out=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO);
//--- Show brief account information
   PrintFormat("The account of the client '%s' #%d %s opened in '%s' on th
               name,login,trade_mode,company,server);
   PrintFormat("Account currency - %s, MarginCall and StopOut levels are s
               currency,(stop_out_mode==ACCOUNT_STOPOUT_MODE_PERCENT)?"per
   PrintFormat("MarginCall=%G, StopOut=%G",margin_call,stop_out);
  }
```

Testing Statistics

After the testing is over, different parameters of the trading results statistics are calculated. The values of the parameters can be obtained using the TesterStatistics() function, by specifying the parameter ID from the ENUM_STATISTICS enumeration.

Although two types of parameters (int and double) are used for calculating statistics, the function returns all values in the double form. All the statistic values of the double type are expressed in the deposit currency by default, unless otherwise specified.

ENUM_STATISTICS

ID	Description of a statistic parameter	Type
STAT_INITIAL_DEPOSIT	The value of the initial deposit	double
STAT_PROFIT	Net profit after testing, the sum of STAT_GROSS_PROFIT and STAT_GROSS_LOSS (STAT_GROSS_LOSS is always less than or equal to zero)	double
STAT_GROSS_PROFIT	Total profit, the sum of all profitable (positive) trades. The value is greater than or equal to zero	double
STAT_GROSS_LOSS	Total loss, the sum of all negative trades. The value is less than or equal to zero	double
STAT_MAX_PROFITTRADE	Maximum profit the largest value of all profitable trades. The value is greater than or equal to zero	double
STAT_MAX_LOSSTRADE	Maximum loss the lowest value of all losing trades. The value is less than or equal to zero	double
STAT_CONPROFITMAX	Maximum profit in a series of profitable trades. The value is greater than or equal to zero	double
STAT_CONPROFITMAX_TRADES	The number of trades that have formed STAT_CONPROFITMAX (maximum profit in a series of profitable trades)	int
STAT_MAX_CONWINS	The total profit of the longest series of	double

	profitable trades	
STAT_MAX_CONPROFIT_TRADES	The number of trades in the longest series of profitable trades STAT_MAX_CONWINS	int
STAT_CONLOSSMAX	Maximum loss in a series of losing trades. The value is less than or equal to zero	double
STAT_CONLOSSMAX_TRADES	The number of trades that have formed STAT_CONLOSSMAX (maximum loss in a series of losing trades)	int
STAT_MAX_CONLOSSES	The total loss of the longest series of losing trades	double
STAT_MAX_CONLOSS_TRADES	The number of trades in the longest series of losing trades STAT_MAX_CONLOSSES	int
STAT_BALANCEMIN	Minimum balance value	double
STAT_BALANCE_DD	Maximum balance drawdown in monetary terms. In the process of trading, a balance may have numerous drawdowns; here the largest value is taken	double
STAT_BALANCEDD_PERCENT	Balance drawdown as a percentage that was recorded at the moment of the maximum balance drawdown in monetary terms (STAT_BALANCE_DD).	double
STAT_BALANCE_DDREL_PERCENT	Maximum balance drawdown as a percentage. In the process of trading, a balance may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	double
STAT_BALANCE_DD_RELATIVE	Balance drawdown in monetary terms that was recorded at the moment of the maximum balance drawdown as a percentage (STAT_BALANCE_DDREL_PERCENT).	double
STAT_EQUITYMIN	Minimum equity value	double
STAT_EQUITY_DD	Maximum equity drawdown in monetary	double

	terms. In the process of trading, numerous drawdowns may appear on the equity; here the largest value is taken	
STAT_EQUITYDD_PERCENT	Drawdown in percent that was recorded at the moment of the maximum equity drawdown in monetary terms (STAT_EQUITY_DD).	double
STAT_EQUITY_DDREL_PERCENT	Maximum equity drawdown as a percentage. In the process of trading, an equity may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	double
STAT_EQUITY_DD_RELATIVE	Equity drawdown in monetary terms that was recorded at the moment of the maximum equity drawdown in percent (STAT_EQUITY_DDREL_PERCENT).	double
STAT_EXPECTED_PAYOFF	Expected payoff	double
STAT_PROFIT_FACTOR	Profit factor, equal to the ratio of STAT_GROSS_PROFIT/STAT_GROSS_LOSS. If STAT_GROSS_LOSS=0, the profit factor is equal to DBL_MAX	double
STAT_MIN_MARGINLEVEL	Minimum value of the margin level	double
STAT_CUSTOM_ONTESTER	The value of the calculated custom optimization criterion returned by the OnTester() function	double
STAT_TRADES	The number of trades	int
STAT_PROFIT_TRADES	Profitable trades	int
STAT_LOSS_TRADES	Losing trades	int
STAT_SHORT_TRADES	Short trades	int
STAT_LONG_TRADES	Long trades	int
STAT_PROFIT_SHORTTRADES	Profitable short trades	int
STAT_PROFIT_LONGTRADES	Profitable long trades	int
STAT_PROFITTRADES_AVGCON	Average length of a profitable series of trades	int
STAT_LOSSTRADES_AVGCON	Average length of a losing series of	int

	trades	

Trade Constants

Various constants used for programming trading strategies are divided into the following groups:

- History Database Properties receiving general information on a symbol;
- Order Properties obtaining information about trade orders;
- Signal Properties - obtaining information about trade signals;

History Database Properties

When accessing timeseries the SeriesInfoInteger() function is used for obtaining additional symbol information. Identifier of a required property is passed as the function parameter. The identifier can be one of values of ENUM_SERIES_INFO_INTEGER.

ENUM_SERIES_INFO_INTEGER

Identifier	Description	Type
SERIES_BARS_COUNT	Bars count for the symbol-period for the current moment	long
SERIES_FIRSTDATE	The very first date for the symbol-period for the current moment	datetime
SERIES_LASTBAR_DATE	Open time of the last bar of the symbol-period	datetime
SERIES_SERVER_FIRSTDATE	The very first date in the history of the symbol on the server regardless of the timeframe	datetime

Order Properties

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

ID	Value	Description
OP_BUY	0	Buy operation
OP_SELL	1	Sell operation
OP_BUYLIMIT	2	Buy limit pending order
OP_SELLLIMIT	3	Sell limit pending order
OP_BUYSTOP	4	Buy stop pending order
OP_SELLSTOP	5	Sell stop pending order

Signal Properties

The following enumerations are used when working with trading signals and signal copy settings.

Enumeration of double type properties of the trading signal:

ENUM_SIGNAL_BASE_DOUBLE

ID	Description
SIGNAL_BASE_BALANCE	Account balance
SIGNAL_BASE_EQUITY	Account equity
SIGNAL_BASE_GAIN	Account gain
SIGNAL_BASE_MAX_DRAWDOWN	Account maximum drawdown
SIGNAL_BASE_PRICE	Signal subscription price
SIGNAL_BASE_ROI	Return on Investment (%)

Enumeration of integer type properties of the trading signal:

ENUM_SIGNAL_BASE_INTEGER

ID	Description
SIGNAL_BASE_DATE_PUBLISHED	Publication date (date when it become available for subscription)
SIGNAL_BASE_DATE_STARTED	Monitoring starting date
SIGNAL_BASE_ID	Signal ID
SIGNAL_BASE_LEVERAGE	Account leverage
SIGNAL_BASE_PIPS	Profit in pips
SIGNAL_BASE_RATING	Position in rating
SIGNAL_BASE_SUBSCRIBERS	Number of subscribers
SIGNAL_BASE_TRADES	Number of trades
SIGNAL_BASE_TRADE_MODE	Account type (0-real, 1-demo, 2-contest)

Enumeration of string type properties of the trading signal:

ENUM_SIGNAL_BASE_STRING

ID	Description
SIGNAL_BASE_AUTHOR_LOGIN	Author login

SIGNAL_BASE_BROKER	Broker name (company)
SIGNAL_BASE_BROKER_SERVER	Broker server
SIGNAL_BASE_NAME	Signal name
SIGNAL_BASE_CURRENCY	Signal base currency

Enumeration of double type properties of the signal copy settings:

ENUM_SIGNAL_INFO_DOUBLE

ID	Description
SIGNAL_INFO_EQUITY_LIMIT	Minimum equity value, below which trade copying is stopped automatically and all orders opened by subscription are closed
SIGNAL_INFO_SLIPPAGE	Allowable subscription when copying deals, in spreads
SIGNAL_INFO_VOLUME_PERCENT	Percentage of volume conversion when copying deals, r/o

Enumeration of integer type properties of the signal copy settings:

ENUM_SIGNAL_INFO_INTEGER

ID	Description
SIGNAL_INFO_CONFIRMATIONS_DISABLED	The flag enables synchronization without confirmation dialog
SIGNAL_INFO_COPY_SLTP	Copy Stop Loss and Take Profit flag
SIGNAL_INFO_DEPOSIT_PERCENT	Percentage of trading account funds used when following providers' signals (in %)
SIGNAL_INFO_ID	Signal id, r/o
SIGNAL_INFO_SUBSCRIPTION_ENABLED	"Copy trades by subscription" permission flag
SIGNAL_INFO_TERMS_AGREE	"Agree to terms of use of Signals service" flag, r/o

Enumeration of string type properties of the signal copy settings:

ENUM_SIGNAL_INFO_STRING

ID	Description
SIGNAL_INFO_NAME	Signal name, r/o

See also

[Trade signals]

Named Constants

All constants used in MQL4 can be divided into the following groups:

- Predefined macro substitutions values are substituted during compilation;
- Mathematical constants values of some mathematical expressions;
- Numerical type constants some of the simple type restrictions;
- Uninitialization reason codes description of uninitialization reasons;
- Checking Object Pointer enumeration of types of pointers returned by the CheckPointer() function;
- Other constants all other constants.

Predefined Macro Substitutions

To simplify the debugging process and obtain information about operation of a mql4-program, there are special macro constant, values of which are set at the moment of compilation. The easiest way to use these constants is outputting values by the Print() function, as it's shown in the example.

Constant	Description
__DATE__	File compilation date without time (hours, minutes and seconds are equal to 0)
__DATETIME__	File compilation date and time
__LINE__	Line number in the source code, in which the macro is located
__FILE__	Name of the currently compiled file
__PATH__	An absolute path to the file that is currently being compiled
__FUNCTION__	Name of the function, in whose body the macro is located
__FUNCSIG__	Signature of the function in whose body the macro is located. Logging of the full description of functions can be useful in the identification of overloaded functions
__MQLBUILD__, __MQL4BUILD__	Compiler build number

Example:

```
#property copyright "Copyright © 2009, MetaQuotes Software Corp." #propert
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- an example of information output at Expert Advisor initialization
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
//--- set the interval between the timer events
   EventSetTimer(5);
//---
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
```

```
//--- an example of information output at Expert Advisor deinitialization
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
//---
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- information output at tick receipt
   Print(" __MQLBUILD__ = ",__MQLBUILD__,"  __FILE__ = ",__FILE__);
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
   test1(__FUNCTION__);
   test2();
//---
  }
//+------------------------------------------------------------------+
//| test1                                                            |
//+------------------------------------------------------------------+
void test1(string par)
  {
//--- information output inside the function
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__," par = "
  }
//+------------------------------------------------------------------+
//| test2                                                            |
//+------------------------------------------------------------------+
void test2()
  {
//--- information output inside the function
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
  }
//+------------------------------------------------------------------+
//| OnTimer event handler                                            |
//+------------------------------------------------------------------+
void OnTimer()
  {
//---
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
   test1(__FUNCTION__);
  }
//+------------------------------------------------------------------+
```

Mathematical Constants

Special constants containing values are reserved for some mathematical expressions. These constants can be used in any place of the program instead of calculating their values using mathematical functions.

Constant	Description	Value
M_E	e	2.71828182845904523536
M_LOG2E	log2(e)	1.44269504088896340736
M_LOG10E	log10(e)	0.434294481903251827651
M_LN2	ln(2)	0.693147180559945309417
M_LN10	ln(10)	2.30258509299404568402
M_PI	pi	3.14159265358979323846
M_PI_2	pi/2	1.57079632679489661923
M_PI_4	pi/4	0.785398163397448309616
M_1_PI	1/pi	0.318309886183790671538
M_2_PI	2/pi	0.636619772367581343076
M_2_SQRTPI	2/sqrt(pi)	1.12837916709551257390
M_SQRT2	sqrt(2)	1.41421356237309504880
M_SQRT1_2	1/sqrt(2)	0.707106781186547524401

Example:

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
void OnStart()
  {
//--- print the values of constants
   Print("M_E = ",DoubleToString(M_E,16));
   Print("M_LOG2E = ",DoubleToString(M_LOG2E,16));
   Print("M_LOG10E = ",DoubleToString(M_LOG10E,16));
   Print("M_LN2 = ",DoubleToString(M_LN2,16));
   Print("M_LN10 = ",DoubleToString(M_LN10,16));
   Print("M_PI = ",DoubleToString(M_PI,16));
   Print("M_PI_2 = ",DoubleToString(M_PI_2,16));
   Print("M_PI_4 = ",DoubleToString(M_PI_4,16));
   Print("M_1_PI = ",DoubleToString(M_1_PI,16));
   Print("M_2_PI = ",DoubleToString(M_2_PI,16));
   Print("M_2_SQRTPI = ",DoubleToString(M_2_SQRTPI,16));
   Print("M_SQRT2 = ",DoubleToString(M_SQRT2,16));
   Print("M_SQRT1_2 = ",DoubleToString(M_SQRT1_2,16));
  }
```

Numerical Type Constants

Each simple numerical type is intended for a certain type of tasks and allows optimizing the operation of a mql4-program when used correctly. For a better code readability and correct handling of calculation results, there are constants which allow to receive information about restrictions set to a certain type of simple data.

Constant	Description	Value
CHAR_MIN	Minimal value, which can be represented by char type	-128
CHAR_MAX	Maximal value, which can be represented by char type	127
UCHAR_MAX	Maximal value, which can be represented by uchar type	255
SHORT_MIN	Minimal value, which can be represented by short type	-32768
SHORT_MAX	Maximal value, which can be represented by short type	32767
USHORT_MAX	Maximal value, which can be represented by ushort type	65535
INT_MIN	Minimal value, which can be represented by int type	-2147483648
INT_MAX	Maximal value, which can be represented by int type	2147483647
UINT_MAX	Maximal value, which can be represented by uint type	4294967295
LONG_MIN	Minimal value, which can be represented by long type	-9223372036854775808
LONG_MAX	Maximal value, which can be represented by long type	9223372036854775807
ULONG_MAX	Maximal value, which can be represented by ulong type	18446744073709551615
DBL_MIN	Minimal positive value, which can be represented by double type	2.2250738585072014e-308
DBL_MAX	Maximal value, which can be	1.7976931348623158e+308

	represented by double type	
DBL_EPSILON	Minimal value, which satisfies the condition: 1.0+DBL_EPSILON != 1.0 (for double type)	2.2204460492503131e-016
DBL_DIG	Number of significant decimal digits for double type	15
DBL_MANT_DIG	Number of bits in a mantissa for double type	53
DBL_MAX_10_EXP	Maximal decimal value of exponent degree for double type	308
DBL_MAX_EXP	Maximal binary value of exponent degree for double type	1024
DBL_MIN_10_EXP	Minimal decimal value of exponent degree for double type	(-307)
DBL_MIN_EXP	Minimal binary value of exponent degree for double type	(-1021)
FLT_MIN	Minimal positive value, which can be represented by float type	1.175494351e-38
FLT_MAX	Maximal value, which can be represented by float type	3.402823466e+38
FLT_EPSILON	Minimal value, which satisfies the condition: 1.0+DBL_EPSILON != 1.0 (for float type)	1.192092896e07
FLT_DIG	Number of significant decimal digits for float type	6
FLT_MANT_DIG	Number of bits in a mantissa for float type	24
FLT_MAX_10_EXP	Maximal decimal value of exponent degree for float type	38
FLT_MAX_EXP	Maximal binary value of exponent degree for float type	128
FLT_MIN_10_EXP	Minimal decimal value of exponent degree for float type	-37
FLT_MIN_EXP	Minimal binary value of exponent degree for float type	(-125)

Example:

```
void OnStart()   {
//--- print the constant values
   printf("CHAR_MIN = %d",CHAR_MIN);
   printf("CHAR_MAX = %d",CHAR_MAX);
   printf("UCHAR_MAX = %d",UCHAR_MAX);
   printf("SHORT_MIN = %d",SHORT_MIN);
   printf("SHORT_MAX = %d",SHORT_MAX);
   printf("USHORT_MAX = %d",USHORT_MAX);
   printf("INT_MIN = %d",INT_MIN);
   printf("INT_MAX = %d",INT_MAX);
   printf("UINT_MAX = %u",UINT_MAX);
   printf("LONG_MIN = %I64d",LONG_MIN);
   printf("LONG_MAX = %I64d",LONG_MAX);
   printf("ULONG_MAX = %I64u",ULONG_MAX);
   printf("EMPTY_VALUE = %.16e",EMPTY_VALUE);
   printf("DBL_MIN = %.16e",DBL_MIN);
   printf("DBL_MAX = %.16e",DBL_MAX);
   printf("DBL_EPSILON = %.16e",DBL_EPSILON);
   printf("DBL_DIG = %d",DBL_DIG);
   printf("DBL_MANT_DIG = %d",DBL_MANT_DIG);
   printf("DBL_MAX_10_EXP =  %d",DBL_MAX_10_EXP);
   printf("DBL_MAX_EXP = %d",DBL_MAX_EXP);
   printf("DBL_MIN_10_EXP = %d",DBL_MIN_10_EXP);
   printf("DBL_MIN_EXP = %d",DBL_MIN_EXP);
   printf("FLT_MIN = %.8e",FLT_MIN);
   printf("FLT_MAX = %.8e",FLT_MAX);
   printf("FLT_EPSILON = %.8e",FLT_EPSILON);
  }
```

Uninitialization Reason Codes

Uninitialization reason codes are returned by the UninitializeReason() function. The possible values are the following:

Constant	Value	Description
REASON_PROGRAM	0	Expert Advisor terminated its operation by calling the ExpertRemove() function
REASON_REMOVE	1	Program has been deleted from the chart
REASON_RECOMPILE	2	Program has been recompiled
REASON_CHARTCHANGE	3	Symbol or chart period has been changed
REASON_CHARTCLOSE	4	Chart has been closed
REASON_PARAMETERS	5	Input parameters have been changed by a user
REASON_ACCOUNT	6	Another account has been activated or reconnection to the trade server has occurred due to changes in the account settings
REASON_TEMPLATE	7	A new template has been applied
REASON_INITFAILED	8	This value means that OnInit() handler has returned a nonzero value
REASON_CLOSE	9	Terminal has been closed

The uninitialization reason code is also passed as a parameter of the predetermined function OnDeinit(const int reason).

The codes 1(REASON_REMOVE) and 2(REASON_RECOMPILE) are implemented for the indicators.

Example:

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
string getUninitReasonText(int reasonCode)
  {
   string text="";
//---
   switch(reasonCode)
     {
      case REASON_ACCOUNT:
         text="Account was changed";break;
      case REASON_CHARTCHANGE:
         text="Symbol or timeframe was changed";break;
      case REASON_CHARTCLOSE:
         text="Chart was closed";break;
      case REASON_PARAMETERS:
         text="Input-parameter was changed";break;
      case REASON_RECOMPILE:
         text="Program "+__FILE__+" was recompiled";break;
      case REASON_REMOVE:
         text="Program "+__FILE__+" was removed from chart";break;
      case REASON_TEMPLATE:
         text="New template was applied to chart";break;
      default:text="Another reason";
     }
//---
   return text;
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- The first way to get the uninitialization reason code
   Print(__FUNCTION__,"_Uninitalization reason code = ",reason);
//--- The second way to get the uninitialization reason code
   Print(__FUNCTION__,"_UninitReason = ",getUninitReasonText(_UninitReason
  }
```

Checking Object Pointer

The CheckPointer() function is used for checking the type of the object pointer. The function returns a value of the ENUM_POINTER_TYPE enumeration. If an incorrect pointer is used, the program execution will be immediately terminated.

Objects created by the new() operator are of POINTER_DYNAMIC type. The delete() operator can and should be used only for such pointers.

All other pointers are of POINTER_AUTOMATIC type, which means that this object has been created automatically by the mql4 program environment. Such objects are deleted automatically after being used.

ENUM_POINTER_TYPE

Constant	Description
POINTER_INVALID	Incorrect pointer
POINTER_DYNAMIC	Pointer of the object created by the new() operator
POINTER_AUTOMATIC	Pointer of any objects created automatically (not using new())

See also

Runtime errors, Object Delete Operator delete, CheckPointer()

Other constants

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

Constant	Description	Value
NULL	Zero for any types. Also indicates empty state of the string	0
EMPTY	Indicates empty state of the parameter	-1
EMPTY_VALUE	Empty value in an indicator buffer. Default custom indicator empty value	2147483647 (0x7FFFFFFF)
CLR_NONE, clrNONE	Absence of color. Indicates empty state of colors	-1
CHARTS_MAX	The maximum possible number of simultaneously open charts in the terminal	100
INVALID_HANDLE	Incorrect handle	-1
IS_DEBUG_MODE	Flag that a mql4-program operates in debug mode	non zero in debug mode, otherwise zero
IS_PROFILE_MODE	Flag that a mql4-program operates in profiling mode	non zero in profiling mode, otherwise zero
WHOLE_ARRAY	Used with array functions. Indicates that all array elements will be processed. Means the number of items remaining until the end of the array, i.e., the entire array will be processed	0
WRONG_VALUE	The constant can be implicitly cast to any enumeration type	-1

The NULL constant can be assigned to a variable of any simple type or to an object structure or class pointer. The NULL assignment for a string variable means the full deinitialization of this variable.

The EMPTY_VALUE constant usually corresponds to the values of indicators that are not shown in the chart. For example, for built-in indicator Standard Deviation with a period of 20, the line for the first 19 bars in the history are not shown in the chart.

The CLR_NONE constant is used to outline the absence of color, it means that the graphical object or graphical series of an indicator will not be plotted. This constant was not included into the Web-color constants list, but it can be applied everywhere where the color arguments are required.

The INVALID_HANDLE constant can be used for checking file handles (see FileOpen() and FileFindFirst()).

The WRONG_VALUE constant is intended for cases, when it is necessary to return value of an enumeration, and this must be a wrong value. For example, when we need to inform that a return value is a value from this enumeration. Let's consider as an example some function CheckLineStyle(), which returns the line style for an object, specified by its name. If at style check by ObjectGetInteger() the result is true, a value from ENUM_LINE_STYLE is returned; otherwise WRONG_VALUE is returned.

```
void OnStart()   {
   if(CheckLineStyle("MyChartObject")==WRONG_VALUE)
      printf("Error line style getting.");
  }
//+------------------------------------------------------------------+
//| returns the line style for an object specified by its name       |
//+------------------------------------------------------------------+
ENUM_LINE_STYLE CheckLineStyle(string name)
  {
   long style;
//---
   if(ObjectGetInteger(0,name,OBJPROP_STYLE,0,style))
      return((ENUM_LINE_STYLE)style);
   else
      return(WRONG_VALUE);
  }
```

The WHOLE_ARRAY constant is intended for functions that require specifying the number of elements in processed arrays:

· ArrayCopy();

· ArrayMinimum();

· ArrayMaximum();

- FileReadArray();
- FileWriteArray().

If you want to specify that all the array values from a specified position till the end must be processed, you should specify just the WHOLE_ARRAY value.

IS_PROFILE_MODE constant allows changing a program operation for correct data collection in the profiling mode. Profiling allows measuring the execution time of the individual program fragments (usually comprising functions), as well as calculating the number of such calls. Sleep() function calls can be disabled to determine the execution time in the profiling mode, like in this example:

```
//--- Sleep can greatly affect (change) profiling result
if(!IS_PROFILE_MODE) Sleep(100); // disabling Sleep() call in the profilin
```

IS_PROFILE_MODE constant value is set by the compiler during the compilation, while it is set to zero in conventional mode. When launching a program in the profiling mode, a special compilation is performed and IS_PROFILE_MODE is replaced with a non-zero value.

The IS_DEBUG_MODE constant can be useful when you need to slightly change the operation of a mql4 program in the debugging mode. For example, in debug mode you may need to display additional debugging information in the terminal log or create additional graphical objects in a chart.

The following example creates a Label object and sets its description and color depending on the script running mode. In order to run a script in the debug mode from MetaEditor, press **F5**. If you run the script from the browser window in the terminal, then the color and text of the object Label will be different.

Example:

```
//+------------------------------------------------------------------+
//|                                             Check_DEBUG_MODE.mq5 |
//|                      Copyright © 2009, MetaQuotes Software Corp. |
//|                                        http://www.metaquotes.net |
//+------------------------------------------------------------------+
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net"
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   string label_name="invisible_label";
   if(ObjectFind(0,label_name)<0)
     {
      Print("Object",label_name,"not found. Error code = ",GetLastError())
      //--- create Label
      ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
      //--- set X coordinate
      ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
      //--- set Y coordinate
      ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
      ResetLastError();
      if(IS_DEBUG_MODE) // debug mode
        {
         //--- show message about the script execution mode
         ObjectSetString(0,label_name,OBJPROP_TEXT,"DEBUG MODE");
         //--- set text color to red
         if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrRed))
            Print("Unable to set the color. Error",GetLastError());
        }
      else              // operation mode
        {
         ObjectSetString(0,label_name,OBJPROP_TEXT,"RELEASE MODE");
         //--- set text color to invisible
         if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,CLR_NONE))
            Print("Unable to set the color. Error ",GetLastError());
        }
      ChartRedraw();
      DebugBreak();    // here termination will occur, if we are in debug
     }
  }
```

Crypt Methods

The ENUM_CRYPT_METHOD enumeration is used to specify the data transformation method, used in CryptEncode() and CryptDecode() functions.

ENUM_CRYPT_METHOD

Constant	Description
CRYPT_BASE64	BASE64
CRYPT_AES128	AES encryption with 128 bit key (16 bytes)
CRYPT_AES256	AES encryption with 256 bit key (32 bytes)
CRYPT_DES	DES encryption with 56 bit key (7 bytes)
CRYPT_HASH_SHA1	SHA1 HASH calculation
CRYPT_HASH_SHA256	SHA256 HASH calculation
CRYPT_HASH_MD5	MD5 HASH calculation
CRYPT_ARCH_ZIP	ZIP archives

See also

DebugBreak(), Executed MQL4 program properties, CryptEncode(), CryptDecode()

Data Structures

MQL4 Language offers 8 predefined structures:

- MqlDateTime is intended for working with date and time;
- MqlRates is intended for manipulating the historical data, it contains information about the price, volume and spread;
- MqlTick is designed for fast retrieval of the most requested information about current prices.

MqlDateTime

The date type structure contains eight fields of the int type:

```
struct MqlDateTime   {
   int year;           // Year
   int mon;            // Month
   int day;            // Day
   int hour;           // Hour
   int min;            // Minutes
   int sec;            // Seconds
   int day_of_week;    // Day of week (0-Sunday, 1-Monday, ... ,6-Saturday
   int day_of_year;    // Day number of the year (January 1st is assigned
  };
```

Note

The day number of the year day_of_year for the leap year, since March, will differ from a number of the corresponding day for a non-leap year.

Example:

```
void OnStart()
  {
//---
   datetime date1=D'2008.03.01';
   datetime date2=D'2009.03.01';

   MqlDateTime str1,str2;
   TimeToStruct(date1,str1);
   TimeToStruct(date2,str2);
   printf("%02d.%02d.%4d, day of year = %d",str1.day,str1.mon,
          str1.year,str1.day_of_year);
   printf("%02d.%02d.%4d, day of year = %d",str2.day,str2.mon,
          str2.year,str2.day_of_year);
  }
/*  Result:
   01.03.2008, day of year = 60
   01.03.2009, day of year = 59
*/
```

See also

TimeToStruct, Structures and Classes

MqlRates

This structure stores information about the prices, volumes and spread.

```
struct MqlRates   {
   datetime time;         // Period start time
   double   open;         // Open price
   double   high;         // The highest price of the period
   double   low;          // The lowest price of the period
   double   close;        // Close price
   long     tick_volume;  // Tick volume
   int      spread;       // Spread
   long     real_volume;  // Trade volume
  };
```

Example:

```
void OnStart()
  {
   MqlRates rates[];
   int copied=CopyRates(NULL,0,0,100,rates);
   if(copied<=0)
      Print("Error copying price data ",GetLastError());
   else Print("Copied ",ArraySize(rates)," bars");
  }
```

See also

CopyRates, Access to timeseries

The Structure for Returning Current Prices (MqlTick)

This is a structure for storing the latest prices of the symbol. It is designed for fast retrieval of the most requested information about current prices.

```
struct MqlTick   {
   datetime     time;          // Time of the last prices update
   double       bid;           // Current Bid price
   double       ask;           // Current Ask price
   double       last;          // Price of the last deal (Last)
   ulong        volume;        // Volume for the current Last price
  };
```

The variable of the MqlTick type allows obtaining values of Ask, Bid, Last and Volume within a single call of the SymbolInfoTick() function.

Example:

```
void OnTick()
  {
   MqlTick last_tick;
//---
   if(SymbolInfoTick(Symbol(),last_tick))
     {
      Print(last_tick.time,": Bid = ",last_tick.bid,
            " Ask = ",last_tick.ask,"  Volume = ",last_tick.volume);
     }
   else Print("SymbolInfoTick() failed, error = ",GetLastError());
//---
  }
```

See also

Structures and Classes

Codes of Errors and Warnings

This section contains the following descriptions:

- Return codes of the trade server analyzing results of the trade request sent by function OrderSend();
- Compiler warnings codes of warning messages that appear at compilation (not errors);
- Compilation errors codes of error messages at an unsuccessful attempt to compile;
- Runtime errors error codes in the execution of mql4-programs, which can be obtained using the GetLastError() function.

Trade Server Return Codes

GetLastError() - returns error codes. Error codes are defined in stderror.mqh. To print the error description you can use the ErrorDescription() function, defined in stdlib.mqh.

Example:

```
#include <stderror.mqh> #include <stdlib.mqh>
void SendMyMessage(string text)
  {
   int check;
   SendMail("Test", text);
   check=GetLastError();
   if(check!=ERR_NO_ERROR) Print("Message not sent. Error: ",ErrorDescript
  }
```

Code	ID	Description
0	ERR_NO_ERROR	No error returned
1	ERR_NO_RESULT	No error returned, but the result is unknown
2	ERR_COMMON_ERROR	Common error
3	ERR_INVALID_TRADE_PARAMETERS	Invalid trade parameters
4	ERR_SERVER_BUSY	Trade server is busy
5	ERR_OLD_VERSION	Old version of the client terminal
6	ERR_NO_CONNECTION	No connection with trade server
7	ERR_NOT_ENOUGH_RIGHTS	Not enough rights
8	ERR_TOO_FREQUENT_REQUESTS	Too frequent requests
9	ERR_MALFUNCTIONAL_TRADE	Malfunctional trade operation
64	ERR_ACCOUNT_DISABLED	Account disabled
65	ERR_INVALID_ACCOUNT	Invalid account
128	ERR_TRADE_TIMEOUT	Trade timeout
129	ERR_INVALID_PRICE	Invalid price
130	ERR_INVALID_STOPS	Invalid stops
131	ERR_INVALID_TRADE_VOLUME	Invalid trade volume

132	ERR_MARKET_CLOSED	Market is closed
133	ERR_TRADE_DISABLED	Trade is disabled
134	ERR_NOT_ENOUGH_MONEY	Not enough money
135	ERR_PRICE_CHANGED	Price changed
136	ERR_OFF_QUOTES	Off quotes
137	ERR_BROKER_BUSY	Broker is busy
138	ERR_REQUOTE	Requote
139	ERR_ORDER_LOCKED	Order is locked
140	ERR_LONG_POSITIONS_ONLY_ALLOWED	Buy orders only allowed
141	ERR_TOO_MANY_REQUESTS	Too many requests
145	ERR_TRADE_MODIFY_DENIED	Modification denied because order is too close to market
146	ERR_TRADE_CONTEXT_BUSY	Trade context is busy
147	ERR_TRADE_EXPIRATION_DENIED	Expirations are denied by broker
148	ERR_TRADE_TOO_MANY_ORDERS	The amount of open and pending orders has reached the limit set by the broker
149	ERR_TRADE_HEDGE_PROHIBITED	An attempt to open an order opposite to the existing one when hedging is disabled
150	ERR_TRADE_PROHIBITED_BY_FIFO	An attempt to close an order contravening the FIFO rule

Compiler Warnings

Compiler warnings are shown for informational purposes only and are not error messages.

Code	Description
21	Incomplete record of a date in the datetime string
22	Wrong number in the datetime string for the date. Requirements: Year 1970 <= X <= 3000 Month 0 <X <= 12 Day 0 <X <= 31/30/28 (29)....
23	Wrong number of datetime string for time. Requirements: Hour 0 <= X <24 Minute 0 <= X <60
24	Invalid color in RGB format: one of RGB components is less than 0 or greater than 255
25	Unknown character of the escape sequences. Known: \n \r \t \\ \" \' \X \x
26	Too large volume of local variables (> 512Kb) of the function, reduce the number
29	Enumeration already defined (duplication) - members will be added to the first definition
30	Overriding macro
31	The variable is declared but is not used anywhere
32	Constructor must be of void type
33	Destructor must be of void type
34	Constant does not fit in the range of integers (X> _UI64_MAX | | X <_I64_MIN) and will be converted to the double type
35	Too long HEX - more than 16 significant characters (senior nibbles are cut)
36	No nibbles in HEX string "0x"
37	No function - nothing to be performed
38	A non-initialized variable is used
41	Function has no body, and is not called
43	Possible loss of data at typecasting. Example: int x = (double) z;

44	Loss of accuracy (of data) when converting a constant. Example: int x = M_PI
45	Difference between the signs of operands in the operations of comparison. Example: (char) c1> (uchar) c2
46	Problems with function importing - declaration of #import is required or import of functions is closed
47	Too large description - extra characters will not be included in the executable file
48	The number of indicator buffers declared is less than required
49	No color to plot a graphical series in the indicator
50	No graphical series to draw the indicator
51	'OnStart' handler function not found in the script
52	'OnStart' handler function is defined with wrong parameters
53	'OnStart' function can be defined only in a script
54	'OnInit' function is defined with wrong parameters
55	'OnInit' function is not used in scripts
56	'OnDeinit' function is defined with wrong parameters
57	'OnDeinit' function is not used in scripts
58	Two 'OnCalculate' functions are defined. OnCalculate () at one price array will be used
59	Overfilling detected when calculating a complex integer constant
60	Probably, the variable is not initialized.
61	This declaration makes it impossible to refer to the local variable declared on the specified line
62	This declaration makes it impossible to refer to the global variable declared on the specified line
63	Cannot be used for static allocated array
64	This variable declaration hides predefined variable
65	The value of the expression is always true/false
66	Using a variable or bool type expression in mathematical operations is unsafe
67	The result of applying the unary minus operator to an unsigned ulong type is undefined
68	The version specified in the #property version property is unacceptable for

	the Market section; the correct format of #property version id "XXX.YYY"
69	Empty controlled statement found
70	Invalid function return type or incorrect parameters during declaration of the event handler function
71	An implicit cast of structures to one type is required
72	This declaration makes direct access to the member of a class declared in the specified string impossible. Access will be possible only with the scope resolution operation ::
73	Binary constant is too big, high-order digits will be truncated
74	Parameter in the method of the inherited class has a different const modifier, the derived function has overloaded the parent function
75	Negative or too large shift value in shift bitwise operation, execution result is undefined
76	Function must return a value
77	void function returns a value
78	Not all control paths return a value
79	Expressions are not allowed on a global scope
80	Check operator precedence for possible error; use parentheses to clarify precedence
81	Two OnCalCulate() are defined. OHLC version will be used
82	Struct has no members, size assigned to 1 byte
83	Return value of the function should be checked
84	Resource indicator is compiled for debugging. That slows down the performance. Please recompile the indicator to increase performance
85	Too great character code in the string, must be in the range 0 to 65535
86	Unrecognized character in the string
87	No indicator window property (setting the display in the main window or a subwindow) is defined. Property #property indicator_chart_window is applied
88	Property ignored, it must be declared on the global scope. The warning is not generated for the following properties: copyright, link, version and strict.

Compilation Errors

MetaEdtior 5 shows error messages about the program errors detected by the built-in compiler during compilation. The list of these errors is given below in table. To compile a source code into an executable one, press **F7**. Programs that contain errors cannot be compiled until the errors identified by the compiler are eliminated.

Code	Description
100	File reading error
101	Error of opening an *. EX4 for writing
103	Not enough free memory to complete compilation
104	Empty syntactic unit unrecognized by compiler
105	Incorrect file name in #include
106	Error accessing a file in #include (probably the file does not exist)
108	Inappropriate name for #define
109	Unknown command of preprocessor (valid #include, #define, #property, #import)
110	Symbol unknown to compiler
111	Function not implemented (description is present, but no body)
112	Double quote (") omitted
113	Opening angle bracket (<) or double quote (") omitted
114	Single quote (') omitted
115	Closing angle bracket ">" omitted
116	Type not specified in declaration
117	No return operator or return is found not in all branches of the implementation
118	Opening bracket of call parameters was expected
119	Error writing EX4
120	Invalid access to an array
121	The function is not of void type and the return operator must return a value
122	Incorrect declaration of the destructor

123	Colon ":" is missing
124	Variable is already declared
125	Variable with such identifier already declared
126	Variable name is too long (> 250 characters)
127	Structure with such identifier already defined
128	Structure is not defined
129	Structure member with the same name already defined
130	No such structure member
131	Breached pairing of brackets
132	Opening parenthesis "(" expected
133	Unbalanced braces (no "}")
134	Difficult to compile (too much branching, internal stack levels are overfilled)
135	Error of file opening for reading
136	Not enough memory to download the source file into memory
137	Variable is expected
138	Reference cannot be initialized
140	Assignment expected (appears at declaration)
141	Opening brace "{" expected
142	Parameter can be a dynamic array only
143	Use of "void" type is unacceptable
144	No pair for ")" or "]", i.e. "(or" [" is absent
145	No pair for "(or" [", i.e. ") "or"] " is absent
146	Incorrect array size
147	Too many parameters (> 64)
149	This token is not expected here
150	Invalid use of operation (invalid operands)
151	Expression of void type not allowed
152	Operator is expected
153	Misuse of break
154	Semicolon ";" expected
155	Comma "," expected

156	Must be a class type, not struct
157	Expression is expected
158	"non HEX character" found in HEX or too long number (number of digits> 511)
159	String-constant has more than 65534 characters
160	Function definition is unacceptable here
161	Unexpected end of program
162	Forward declaration is prohibited for structures
163	Function with this name is already defined and has another return type
164	Function with this name is already defined and has a different set of parameters
165	Function with this name is already defined and implemented
166	Function overload for this call was not found
167	Function with a return value of void type cannot return a value
168	Function is not defined
170	Value is expected
171	In *case* expression only integer constants are valid
172	The value of *case* in this *switch* is already used
173	Integer is expected
174	In #import expression file name is expected
175	Expressions are not allowed on global level
176	Omitted parenthesis ")" before ";"
177	To the left of equality sign a variable is expected
178	The result of expression is not used
179	Declaring of variables is not allowed in *case*
180	Implicit conversion from a string to a number
181	Implicit conversion of a number to a string
182	Ambiguous call of an overloaded function (several overloads fit)
183	Illegal *else* without proper *if*
184	Invalid *case* or *default* without a *switch*
185	Inappropriate use of ellipsis
186	The initializing sequence has more elements than the initialized variable

187	A constant for *case* expected
188	A constant expression required
189	A constant variable cannot be changed
190	Closing bracket or a comma is expected (declaring array member)
191	Enumerator identifier already defined
192	Enumeration cannot have access modifiers (const, extern, static)
193	Enumeration member already declared with a different value
194	There is a variable defined with the same name
195	There is a structure defined with the same name
196	Name of enumeration member expected
197	Integer expression expected
198	Division by zero in constant expression
199	Wrong number of parameters in the function
200	Parameter by reference must be a variable
201	Variable of the same type to pass by reference expected
202	A constant variable cannot be passed by a non-constant reference
203	Requires a positive integer constant
204	Failed to access protected class member
205	Import already defined in another way
208	Executable file not created
209	'OnCalculate' entry point not found for the indicator
210	The continue operation can be used only inside a loop
211	Error accessing private (closed) class member
213	Method of structure or class is not declared
214	Error accessing private (closed) class method
216	Copying of structures with objects is not allowed
218	Index out of array range
219	Array initialization in structure or class declaration not allowed
220	Class constructor cannot have parameters
221	Class destructor can not have parameters
222	Class method or structure with the same name and parameters have already

	been declared
223	Operand expected
224	Class method or structure with the same name exists, but with different parameters (declaration!=implementation)
225	Imported function is not described
226	ZeroMemory() is not allowed for objects with protected members or inheritance
227	Ambiguous call of the overloaded function (exact match of parameters for several overloads)
228	Variable name expected
229	A reference cannot be declared in this place
230	Already used as the enumeration name
232	Class or structure expected
235	Cannot call 'delete' operator to delete the array
236	Operator ' while' expected
237	Operator 'delete' must have a pointer
238	There is 'default' for this 'switch' already
239	Syntax error
240	Escape-sequence can occur only in strings (starts with '\')
241	Array required - square bracket '[' does not apply to an array, or non arrays are passed as array parameters
242	Can not be initialized through the initialization sequence
243	Import is not defined
244	Optimizer error on the syntactic tree
245	Declared too many structures (try to simplify the program)
246	Conversion of the parameter is not allowed
247	Incorrect use of the 'delete' operator
248	It's not allowed to declare a pointer to a reference
249	It's not allowed to declare a reference to a reference
250	It's not allowed to declare a pointer to a pointer
251	Structure declaration in the list of parameter is not allowed

252	Invalid operation of typecasting
253	A pointer can be declared only for a class or structure
256	Undeclared identifier
257	Executable code optimizer error
258	Executable code generation error
260	Invalid expression for the 'switch' operator
261	Pool of string constants overfilled, simplify program
262	Cannot convert to enumeration
263	Do not use 'virtual' for data (members of a class or structure)
264	Cannot call protected method of class
265	Overridden virtual functions return a different type
266	Class cannot be inherited from a structure
267	Structure cannot be inherited from a class
268	Constructor cannot be virtual (*virtual* specifier is not allowed)
269	Method of structure cannot be virtual
270	Function must have a body
271	Overloading of system functions (terminal functions) is prohibited
272	*Const* specifier is invalid for functions that are not members of a class or structure
274	Not allowed to change class members in constant method
276	Inappropriate initialization sequence
277	Missed default value for the parameter (specific declaration of default parameters)
278	Overriding the default parameter (different values in declaration and implementation)
279	Not allowed to call non-constant method for a constant object
280	An object is necessary for accessing members (a dot for a non class/structure is set)
281	The name of an already declared structure cannot be used in declaration
284	Unauthorized conversion (at closed inheritance)
285	Structures and arrays cannot be used as input variables
286	*Const* specifier is not valid for constructor/destructor

287	Incorrect string expression for a datetime
288	Unknown property (#property)
289	Incorrect value of a property
290	Invalid index for a property in #property
291	Call parameter omitted - <func (x,)>
293	Object must be passed by reference
294	Array must be passed by reference
295	Function was declared as exportable
296	Function was not declared as exportable
297	It is prohibited to export imported function
298	Imported function cannot have this parameter (prohibited to pass a pointer, class or structure containing a dynamic array, pointer, class, etc.)
299	Must be a class
300	#import was not closed
302	Type mismatch
303	Extern variable is already initialized
304	No exported function or entry point found
305	Explicit constructor call is not allowed
306	Method was declared as constant
307	Method was not declared as constant
308	Incorrect size of the resource file
309	Incorrect resource name
310	Resource file opening error
311	Resource file reading error
312	Unknown resource type
313	Incorrect path to the resource file
314	The specified resource name is already used
315	Argument expected for the function-like macro
316	Unexpected symbol in macro definition
317	Error in formal parameters of the macro
318	Invalid number of parameters for a macro

319	Too many parameters for a macro
320	Too complex, simplify the macro
321	Parameter for EnumToString() can be only an enumeration
322	The resource name is too long
323	Unsupported image format (only BMP with 24 or 32 bit color depth is supported)
324	An array cannot be declared in operator
325	The function can be declared only in the global scope
326	The declaration is not allowed for the current scope
327	Initialization of static variables with the values of local variables is not allowed
328	Illegal declaration of an array of objects that do not have a default constructor
329	Initialization list allowed only for constructors
330	No function definition after initialization list
331	Initialization list is empty
332	Array initialization in a constructor is not allowed
333	Initializing members of a parent class in the initialization list is not allowed
334	Expression of the integer type expected
335	Memory required for the array exceeds the maximum value
336	Memory required for the structure exceeds the maximum value
337	Memory required for the variables declared on the global level exceeds the maximum value
338	Memory required for local variables exceeds the maximum value
339	Constructor not defined
340	Invalid name of the icon file
341	Could not open the icon file at the specified path
342	The icon file is incorrect and is not of the ICO format
343	Reinitialization of a member in a class/structure constructor using the initialization list
344	Initialization of static members in the constructor initialization list is not allowed

345	Initialization of a non-static member of a class/structure on a global level is not allowed
346	The name of the class/structure method matches the name of an earlier declared member
347	The name of the class/structure member matches the name of an earlier declared method
348	Virtual function cannot be declared as static
349	The const modifier is not allowed for static functions
350	Constructor or destructor cannot be static
351	Non-static member/method of a class or a structure cannot be accessed from a static function
352	An overload operation (+,-,[],++,-- etc.) is expected after the operator keyword
353	Not all operations can be overloaded in MQL4
354	Definition does not match declaration
355	An invalid number of parameters is specified for the operator
356	Event handling function not found
357	Method cannot be exported
358	A pointer to the constant object cannot be normalized by a non-constant object
359	Class templates are not supported yet
360	Function template overload is not supported yet
361	Function template cannot be applied
362	Ambiguous parameter in function template (several parameter types can be applied)
363	Unable to determine the parameter type, by which the function template argument should be normalized
364	Incorrect number of parameters in the function template
365	Function template cannot be virtual
366	Function templates cannot be exported
367	Function templates cannot be imported
368	Structures containing the objects are not allowed
369	String arrays and structures containing the objects are not allowed

370	A static class/structure member must be explicitly initialized
371	Compiler limitation: the string cannot contain more than 65 535 characters
372	Inconsistent #ifdef/#endif
373	Object of class cannot be returned, copy constructor not found
374	Non-static members and methods cannot be used
375	OnTesterInit() impossible to use without OnTesterDeinit()
376	Redefinition of formal parameter '%s'
377	Macro __FUNCSIG__ and __FUNCTION__ cannot appear outside of a function body
378	Invalid returned type. For example, this error will be produced for functions imported from DLL that return structure or pointer.
379	Template usage error
380	Not used
381	Illegal syntax when declaring pure virtual function, only "=NULL" or "=0" are allowed
382	Only virtual functions can be declared with the pure-specifier ("=NULL" or "=0")
383	Abstract class cannot be instantiated
384	A pointer to a user-defined type should be applied as a target type for dynamic casting using the dynamic_cast operator
385	"Pointer to function" type is expected
386	Pointers to methods are not supported
387	Error cannot define the type of a pointer to function
388	Type cast is not available due to private inheritance
389	A variable with const modifier should be initialized during declaration

Runtime Errors

The GetLastError() function return last error code, stored in _LastError predefined variable. This value can be reset using the ResetLastError() function. Error code constants defined at stderror.mqh file. To print text messages use ErrorDescription() function defined at stdlib.mqh file.

For convenience, trade errors are additionally listed in the Trade Server Return Codes section.

Runtime errors of mql4-program:

Code	ID	Description
0	ERR_NO_ERROR	No error returned
1	ERR_NO_RESULT	No error returned, but the result is unknown
2	ERR_COMMON_ERROR	Common error
3	ERR_INVALID_TRADE_PARAMETERS	Invalid trade parameters
4	ERR_SERVER_BUSY	Trade server is busy
5	ERR_OLD_VERSION	Old version of the client terminal
6	ERR_NO_CONNECTION	No connection with trade server
7	ERR_NOT_ENOUGH_RIGHTS	Not enough rights
8	ERR_TOO_FREQUENT_REQUESTS	Too frequent requests
9	ERR_MALFUNCTIONAL_TRADE	Malfunctional trade operation
64	ERR_ACCOUNT_DISABLED	Account disabled
65	ERR_INVALID_ACCOUNT	Invalid account
128	ERR_TRADE_TIMEOUT	Trade timeout
129	ERR_INVALID_PRICE	Invalid price
130	ERR_INVALID_STOPS	Invalid stops
131	ERR_INVALID_TRADE_VOLUME	Invalid trade volume
132	ERR_MARKET_CLOSED	Market is closed
133	ERR_TRADE_DISABLED	Trade is disabled
134	ERR_NOT_ENOUGH_MONEY	Not enough money
135	ERR_PRICE_CHANGED	Price changed

136	ERR_OFF_QUOTES	Off quotes
137	ERR_BROKER_BUSY	Broker is busy
138	ERR_REQUOTE	Requote
139	ERR_ORDER_LOCKED	Order is locked
140	ERR_LONG_POSITIONS_ONLY_ALLOWED	Buy orders only allowed
141	ERR_TOO_MANY_REQUESTS	Too many requests
145	ERR_TRADE_MODIFY_DENIED	Modification denied because order is too close to market
146	ERR_TRADE_CONTEXT_BUSY	Trade context is busy
147	ERR_TRADE_EXPIRATION_DENIED	Expirations are denied by broker
148	ERR_TRADE_TOO_MANY_ORDERS	The amount of open and pending orders has reached the limit set by the broker
149	ERR_TRADE_HEDGE_PROHIBITED	An attempt to open an order opposite to the existing one when hedging is disabled
150	ERR_TRADE_PROHIBITED_BY_FIFO	An attempt to close an order contravening the FIFO rule
4000	ERR_NO_MQLERROR	No error returned
4001	ERR_WRONG_FUNCTION_POINTER	Wrong function pointer
4002	ERR_ARRAY_INDEX_OUT_OF_RANGE	Array index is out of range
4003	ERR_NO_MEMORY_FOR_CALL_STACK	No memory for function call stack
4004	ERR_RECURSIVE_STACK_OVERFLOW	Recursive stack overflow
4005	ERR_NOT_ENOUGH_STACK_FOR_PARAM	Not enough stack for parameter
4006	ERR_NO_MEMORY_FOR_PARAM_STRING	No memory for parameter string
4007	ERR_NO_MEMORY_FOR_TEMP_STRING	No memory for temp string
4008	ERR_NOT_INITIALIZED_STRING	Not initialized string
4009	ERR_NOT_INITIALIZED_ARRAYSTRING	Not initialized string in array
4010	ERR_NO_MEMORY_FOR_ARRAYSTRING	No memory for array string
4011	ERR_TOO_LONG_STRING	Too long string
4012	ERR_REMAINDER_FROM_ZERO_DIVIDE	Remainder from zero divide
4013	ERR_ZERO_DIVIDE	Zero divide
4014	ERR_UNKNOWN_COMMAND	Unknown command

4015	ERR_WRONG_JUMP	Wrong jump (never generated error)
4016	ERR_NOT_INITIALIZED_ARRAY	Not initialized array
4017	ERR_DLL_CALLS_NOT_ALLOWED	DLL calls are not allowed
4018	ERR_CANNOT_LOAD_LIBRARY	Cannot load library
4019	ERR_CANNOT_CALL_FUNCTION	Cannot call function
4020	ERR_EXTERNAL_CALLS_NOT_ALLOWED	Expert function calls are not allowed
4021	ERR_NO_MEMORY_FOR_RETURNED_STR	Not enough memory for temp string returned from function
4022	ERR_SYSTEM_BUSY	System is busy (never generated error)
4023	ERR_DLLFUNC_CRITICALERROR	DLL-function call critical error
4024	ERR_INTERNAL_ERROR	Internal error
4025	ERR_OUT_OF_MEMORY	Out of memory
4026	ERR_INVALID_POINTER	Invalid pointer
4027	ERR_FORMAT_TOO_MANY_FORMATTERS	Too many formatters in the format function
4028	ERR_FORMAT_TOO_MANY_PARAMETERS	Parameters count exceeds formatters count
4029	ERR_ARRAY_INVALID	Invalid array
4030	ERR_CHART_NOREPLY	No reply from chart
4050	ERR_INVALID_FUNCTION_PARAMSCNT	Invalid function parameters count
4051	ERR_INVALID_FUNCTION_PARAMVALUE	Invalid function parameter value
4052	ERR_STRING_FUNCTION_INTERNAL	String function internal error
4053	ERR_SOME_ARRAY_ERROR	Some array error
4054	ERR_INCORRECT_SERIESARRAY_USING	Incorrect series array using
4055	ERR_CUSTOM_INDICATOR_ERROR	Custom indicator error
4056	ERR_INCOMPATIBLE_ARRAYS	Arrays are incompatible
4057	ERR_GLOBAL_VARIABLES_PROCESSING	Global variables processing error
4058	ERR_GLOBAL_VARIABLE_NOT_FOUND	Global variable not found
4059	ERR_FUNC_NOT_ALLOWED_IN_TESTING	Function is not allowed in testing mode

4060	ERR_FUNCTION_NOT_CONFIRMED	Function is not allowed for call
4061	ERR_SEND_MAIL_ERROR	Send mail error
4062	ERR_STRING_PARAMETER_EXPECTED	String parameter expected
4063	ERR_INTEGER_PARAMETER_EXPECTED	Integer parameter expected
4064	ERR_DOUBLE_PARAMETER_EXPECTED	Double parameter expected
4065	ERR_ARRAY_AS_PARAMETER_EXPECTED	Array as parameter expected
4066	ERR_HISTORY_WILL_UPDATED	Requested history data is in updating state
4067	ERR_TRADE_ERROR	Internal trade error
4068	ERR_RESOURCE_NOT_FOUND	Resource not found
4069	ERR_RESOURCE_NOT_SUPPORTED	Resource not supported
4070	ERR_RESOURCE_DUPLICATED	Duplicate resource
4071	ERR_INDICATOR_CANNOT_INIT	Custom indicator cannot initialize
4072	ERR_INDICATOR_CANNOT_LOAD	Cannot load custom indicator
4073	ERR_NO_HISTORY_DATA	No history data
4074	ERR_NO_MEMORY_FOR_HISTORY	No memory for history data
4075	ERR_NO_MEMORY_FOR_INDICATOR	Not enough memory for indicator calculation
4099	ERR_END_OF_FILE	End of file
4100	ERR_SOME_FILE_ERROR	Some file error
4101	ERR_WRONG_FILE_NAME	Wrong file name
4102	ERR_TOO_MANY_OPENED_FILES	Too many opened files
4103	ERR_CANNOT_OPEN_FILE	Cannot open file
4104	ERR_INCOMPATIBLE_FILEACCESS	Incompatible access to a file
4105	ERR_NO_ORDER_SELECTED	No order selected
4106	ERR_UNKNOWN_SYMBOL	Unknown symbol
4107	ERR_INVALID_PRICE_PARAM	Invalid price
4108	ERR_INVALID_TICKET	Invalid ticket
4109	ERR_TRADE_NOT_ALLOWED	Trade is not allowed. Enable checkbox "Allow live trading" in the Expert Advisor properties

4110	ERR_LONGS_NOT_ALLOWED	Longs are not allowed. Check the Expert Advisor properties
4111	ERR_SHORTS_NOT_ALLOWED	Shorts are not allowed. Check the Expert Advisor properties
4112	ERR_TRADE_EXPERT_DISABLED_BY_SERVER	Automated trading by Expert Advisors/Scripts disabled by trade server
4200	ERR_OBJECT_ALREADY_EXISTS	Object already exists
4201	ERR_UNKNOWN_OBJECT_PROPERTY	Unknown object property
4202	ERR_OBJECT_DOES_NOT_EXIST	Object does not exist
4203	ERR_UNKNOWN_OBJECT_TYPE	Unknown object type
4204	ERR_NO_OBJECT_NAME	No object name
4205	ERR_OBJECT_COORDINATES_ERROR	Object coordinates error
4206	ERR_NO_SPECIFIED_SUBWINDOW	No specified subwindow
4207	ERR_SOME_OBJECT_ERROR	Graphical object error
4210	ERR_CHART_PROP_INVALID	Unknown chart property
4211	ERR_CHART_NOT_FOUND	Chart not found
4212	ERR_CHARTWINDOW_NOT_FOUND	Chart subwindow not found
4213	ERR_CHARTINDICATOR_NOT_FOUND	Chart indicator not found
4220	ERR_SYMBOL_SELECT	Symbol select error
4250	ERR_NOTIFICATION_ERROR	Notification error
4251	ERR_NOTIFICATION_PARAMETER	Notification parameter error
4252	ERR_NOTIFICATION_SETTINGS	Notifications disabled
4253	ERR_NOTIFICATION_TOO_FREQUENT	Notification send too frequent
4260	ERR_FTP_NOSERVER	FTP server is not specified
4261	ERR_FTP_NOLOGIN	FTP login is not specified
4262	ERR_FTP_CONNECT_FAILED	FTP connection failed
4263	ERR_FTP_CLOSED	FTP connection closed
4264	ERR_FTP_CHANGEDIR	FTP path not found on server
4265	ERR_FTP_FILE_ERROR	File not found in the MQL4\Files directory to send on FTP server
4266	ERR_FTP_ERROR	Common error during FTP data transmission

5001	ERR_FILE_TOO_MANY_OPENED	Too many opened files
5002	ERR_FILE_WRONG_FILENAME	Wrong file name
5003	ERR_FILE_TOO_LONG_FILENAME	Too long file name
5004	ERR_FILE_CANNOT_OPEN	Cannot open file
5005	ERR_FILE_BUFFER_ALLOCATION_ERROR	Text file buffer allocation error
5006	ERR_FILE_CANNOT_DELETE	Cannot delete file
5007	ERR_FILE_INVALID_HANDLE	Invalid file handle (file closed or was not opened)
5008	ERR_FILE_WRONG_HANDLE	Wrong file handle (handle index is out of handle table)
5009	ERR_FILE_NOT_TOWRITE	File must be opened with FILE_WRITE flag
5010	ERR_FILE_NOT_TOREAD	File must be opened with FILE_READ flag
5011	ERR_FILE_NOT_BIN	File must be opened with FILE_BIN flag
5012	ERR_FILE_NOT_TXT	File must be opened with FILE_TXT flag
5013	ERR_FILE_NOT_TXTORCSV	File must be opened with FILE_TXT or FILE_CSV flag
5014	ERR_FILE_NOT_CSV	File must be opened with FILE_CSV flag
5015	ERR_FILE_READ_ERROR	File read error
5016	ERR_FILE_WRITE_ERROR	File write error
5017	ERR_FILE_BIN_STRINGSIZE	String size must be specified for binary file
5018	ERR_FILE_INCOMPATIBLE	Incompatible file (for string arrays-TXT, for others-BIN)
5019	ERR_FILE_IS_DIRECTORY	File is directory not file
5020	ERR_FILE_NOT_EXIST	File does not exist
5021	ERR_FILE_CANNOT_REWRITE	File cannot be rewritten
5022	ERR_FILE_WRONG_DIRECTORYNAME	Wrong directory name
5023	ERR_FILE_DIRECTORY_NOT_EXIST	Directory does not exist

5024	ERR_FILE_NOT_DIRECTORY	Specified file is not directory
5025	ERR_FILE_CANNOT_DELETE_DIRECTORY	Cannot delete directory
5026	ERR_FILE_CANNOT_CLEAN_DIRECTORY	Cannot clean directory
5027	ERR_FILE_ARRAYRESIZE_ERROR	Array resize error
5028	ERR_FILE_STRINGRESIZE_ERROR	String resize error
5029	ERR_FILE_STRUCT_WITH_OBJECTS	Structure contains strings or dynamic arrays
5200	ERR_WEBREQUEST_INVALID_ADDRESS	Invalid URL
5201	ERR_WEBREQUEST_CONNECT_FAILED	Failed to connect to specified URL
5202	ERR_WEBREQUEST_TIMEOUT	Timeout exceeded
5203	ERR_WEBREQUEST_REQUEST_FAILED	HTTP request failed
	User errors	
65536	ERR_USER_ERROR_FIRST	User defined errors start with this code

Input and Output Constants

Constants:

· File opening flags
· File properties
· Positioning inside a file
· Code page usage
· MessageBox

File Opening Flags

File opening flag values specify the file access mode. Flags are defined as follows:

Identifier	Value	Description
FILE_READ	1	File is opened for reading. Flag is used in FileOpen(). When opening a file specification of FILE_WRITE and/or FILE_READ is required.
FILE_WRITE	2	File is opened for writing. Flag is used in FileOpen(). When opening a file specification of FILE_WRITE and/or FILE_READ is required.
FILE_BIN	4	Binary read/write mode (without string to string conversion). Flag is used in FileOpen()
FILE_CSV	8	CSV file (all its elements are converted to strings of the appropriate type, unicode or ansi, and separated by separator). Flag is used in FileOpen()
FILE_TXT	16	Simple text file (the same as csv file, but without taking into account the separators). Flag is used in FileOpen()
FILE_ANSI	32	Strings of ANSI type (one byte symbols). Flag is used in FileOpen()
FILE_UNICODE	64	Strings of UNICODE type (two byte symbols). Flag is used in FileOpen()
FILE_SHARE_READ	128	Shared access for reading from several programs. Flag is used in FileOpen(), but it does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.
FILE_SHARE_WRITE	256	Shared access for writing from several programs. Flag is used in FileOpen(), but it does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.
FILE_REWRITE	512	Possibility for the file rewrite using functions FileCopy() and FileMove(). The file should exist or should be opened for writing, otherwise the file will not be opened.
FILE_COMMON	4096	The file path in the common folder of all client terminals \Terminal\Common\Files. Flag is used in FileOpen(), FileCopy(), FileMove() and in FileIsExist() functions.

One or several flags can be specified when opening a file. This is a combination of flags. The combination of flags is written using the sign of logical OR (|), which is positioned between enumerated flags. For example, to open a file in CSV format for reading and writing at the same time, specify the combination FILE_READ|FILE_WRITE|FILE_CSV.

Example:

```
   int filehandle=FileOpen(filename,FILE_READ|FILE_WRITE|FILE_CSV);
```

There are some specific features of work when you specify read and write flags:

- If FILE_READ is specified, an attempt is made to open an existing file. If a file does not exist, file opening fails, a new file is not created.
- FILE_READ|FILE_WRITE a new file is created if the file with the specified name does not exist.
- FILE_WRITE the file is created again with a zero size.

When opening a file, specification of FILE_WRITE and/or FILE_READ is required.

Flags that define the type of reading of an open file possess priority. The highest flag is FILE_CSV, then goes FILE_BIN, and FILE_TXT is of lowest priority. Thus, if several flags are specified at the same time, (FILE_TXT|FILE_CSV or FILE_TXT|FILE_BIN or FILE_BIN|FILE_CSV), the flag with the highest priority will be used.

Flags that define the type of encoding also have priority. FILE_UNICODE is of a higher priority than FILE_ANSI. So if you specify combination FILE_UNICODE|FILE_ANSI, flag FILE_UNICODE will be used.

If neither FILE_UNICODE nor FILE_ANSI is indicated, FILE_UNICODE is implied. If neither FILE_CSV, nor FILE_BIN, nor FILE_TXT is specified, FILE_CSV is implied.

If a file is opened for reading as a text file (FILE_TXT or FILE_CSV), and at the file beginning a special two-byte indication 0xff,0xfe is found, the encoding flag will be FILE_UNICODE, even if FILE_ANSI is specified.

See also

File Functions

File Properties

The FileGetInteger() function is used for obtaining file properties. The identifier of the required property from the ENUM_FILE_PROPERTY_INTEGER enumeration is passed to it during call.

ENUM_FILE_PROPERTY_INTEGER

ID	ID description
FILE_EXISTS	Check the existence
FILE_CREATE_DATE	Date of creation
FILE_MODIFY_DATE	Date of the last modification
FILE_ACCESS_DATE	Date of the last access to the file
FILE_SIZE	File size in bytes
FILE_POSITION	Position of a pointer in the file
FILE_END	Get the end of file sign
FILE_LINE_END	Get the end of line sign
FILE_IS_COMMON	The file is opened in a shared folder of all terminals (see FILE_COMMON)
FILE_IS_TEXT	The file is opened as a text file (see FILE_TXT)
FILE_IS_BINARY	The file is opened as a binary file (see FILE_BIN)
FILE_IS_CSV	The file is opened as CSV (see FILE_CSV)
FILE_IS_ANSI	The file is opened as ANSI (see FILE_ANSI)
FILE_IS_READABLE	The opened file is readable (see FILE_READ)
FILE_IS_WRITABLE	The opened file is writable (see FILE_WRITE)

The FileGetInteger() function has two different options of call. In the first option, for getting properties of a file, its handle is specified, which is obtained while opening the file using the FileOpen() function. This option allows getting all properties of a file.

The second option of the FileGetInteger() function returns values of file properties by the file name. Using this option, only the following general properties can be obtained:

- FILE_EXISTS existence of a file with a specified name
- FILE_CREATE_DATE date of creation of the file with the specified name

- FILE_MODIFY_DATE date of modification of the file with the specified name
- FILE_ACCESS_DATE date of the last access to the file with the specified name
- FILE_SIZE size of the file with the specified name

When trying to get properties other than specified above, the second option of FileGetInteger() call will return an error.

Positioning Inside a File

Most of file functions are associated with data read/write operations. At the same time, using the FileSeek() you can specify the position of a file pointer to a position inside the file, from which the next read or write operation will be performed. The ENUM_FILE_POSITION enumeration contains valid pointer positions, relative to which you can specify the shift in bytes for the next operation.

ENUM_FILE_POSITION

Identifier	Description
SEEK_SET	File beginning
SEEK_CUR	Current position of a file pointer
SEEK_END	File end

See also

FileIsEnding, FileIsLineEnding

Using a Codepage in String Conversion Operations

When converting string variables into arrays of char type and back, the encoding that by default corresponds to the current ANSI of Windows operating system (CP_ACP) is used in MQL4. If you want to specify a different type of encoding, it can be set as additional parameter for the CharArrayToString(), StringToCharArray() and FileOpen() functions.

The table lists the built-in constants for some of the most popular code pages. Not mentioned code pages can be specified by a code corresponding to the page.

Built-in Constants of Codepages

Constant	Value	Description
CP_ACP	0	The current Windows ANSI code page.
CP_OEMCP	1	The current system OEM code page.
CP_MACCP	2	The current system Macintosh code page. **Note:** This value is mostly used in earlier created program codes and is of no use now, since modern Macintosh computers use Unicode for encoding.
CP_THREAD_ACP	3	The Windows ANSI code page for the current thread.
CP_SYMBOL	42	Symbol code page
CP_UTF7	65000	UTF-7 code page.
CP_UTF8	65001	UTF-8 code page.

See also

Client Terminal Properties

Constants of the MessageBox Dialog Window

This section contains return codes of the MessageBox() function. If a message window has a Cancel button, the function returns IDCANCEL, in case if the ESC key or the Cancel button is pressed. If there is no Cancel button in the message window, the pressing of ESC does not give any effect.

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

The main flags of the MessageBox() function define contents and behavior of the dialog window. This value can be a combination of the following flag groups:

Constant	Value	Description
MB_OK	0x00000000	Message window contains only one button: OK. Default
MB_OKCANCEL	0x00000001	Message window contains two buttons: OK and Cancel
MB_ABORTRETRYIGNORE	0x00000002	Message window contains three buttons: Abort, Retry and Ignore
MB_YESNOCANCEL	0x00000003	Message window contains three buttons: Yes, No and Cancel
MB_YESNO	0x00000004	Message window contains two buttons: Yes and No
MB_RETRYCANCEL	0x00000005	Message window contains two buttons: Retry

		and Cancel
MB_CANCELTRYCONTINUE	0x00000006	Message window contains three buttons: Cancel, Try Again, Continue

To display an icon in the message window it is necessary to specify additional flags:

Constant	Value	Description
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	0x00000010	The STOP sign icon
MB_ICONQUESTION	0x00000020	The question sign icon
MB_ICONEXCLAMATION, MB_ICONWARNING	0x00000030	The exclamation/warning sign icon
MB_ICONINFORMATION, MB_ICONASTERISK	0x00000040	The encircled **i** sign

Default buttons are defined by the following flags:

Constant	Value	Description
MB_DEFBUTTON1	0x00000000	The first button MB_DEFBUTTON1 - is default, if the other buttons MB_DEFBUTTON2, MB_DEFBUTTON3, or MB_DEFBUTTON4 are not specified
MB_DEFBUTTON2	0x00000100	The second button is default
MB_DEFBUTTON3	0x00000200	The third button is default
MB_DEFBUTTON4	0x00000300	The fourth button is default

MQL4 Programs

For the mql4-program to operate, it must be compiled (Compile button or F7 key). Compilation should pass without errors (some warnings are possible; they should be analyzed). At this process, an executable file with the same name and with EX4 extension must be created in the corresponding directory, terminal_dir\MQL4\Experts, terminal_dir\MQL4\indicators or terminal_dir\MQL4\scripts. This file can be run.

Operating features of MQL4 programs are described in the following sections:

- Program running order of calling predefined event-handlers.
- Client terminal events description of events, which can be processed in programs.
- Call of imported functions description order, allowed parameters, search details and call agreement for imported functions.
- Runtime errors getting information about runtime and critical errors.

Expert Advisors, custom indicators and scripts are attached to one of opened charts by Drag'n'Drop method from the Navigator window.

For an expert Advisor to stop operating, it should be removed from a chart. To do it select "Expert list" in chart context menu, then select an Expert Advisor from list and click "Remove" button. Operation of Expert Advisors is also affected by the state of the "AutoTrading" button.

In order to stop a custom indicator, it should be removed from a chart.

Custom indicators and Expert Advisors work until they are explicitly removed from a chart; information about attached Expert Advisors and Indicators is saved between client terminal sessions.

Scripts are executed once and are deleted automatically upon operation completion or change of the current chart state, or upon client terminal shutdown. After the restart of the client terminal scripts are not started, because the information about them is not saved.

Maximum one Expert Advisor, one script and unlimited number of indicators can operate in one chart.

Program Running

Each script and each Expert Advisor runs in its own separate thread. All indicators work in the graphic interface thread. processing of ticks and history synchronization are also performed in graphic interface thread. Custom indicators work in the main interface thread. If a custom indicator has been called with the iCustom() function, this indicator works in the thread of the program that has called it. Library (imported) functions work in the calling program thread, as well.

When running an Expert Advisor, make sure that it has an actual trading environment and can access the history of the required symbol and period, and synchronize data between the terminal and the server. For all these procedures, the terminal provides a start delay of no more than 5 seconds, after which the Expert Advisor will be started with available data. Therefore, in case there is no connection to the server, this may lead to a delay in the start of an Expert Advisor.

The below table contains a brief summary of MQL4 programs:

Program	Running	Note
Script	A separate thread, the number of threads for scripts is equal to the number of scripts	A looped script cannot break running of other programs
Expert Advisor	A separate thread, the number of threads for Expert Advisors is equal to the number of Expert Advisors	A looped Expert Advisor cannot break running of other programs
Indicator	All indicators share the resources of graphic interface thread of the terminal	An infinite loop in one indicator will stop the work of terminal

Right after a program is attached to a chart, it is uploaded to the client terminal memory, as well as global variable are initialized. If some global variable of the class type has a constructor, this constructor will be called during initialization of global variables.

After that the program is waiting for an event from the client terminal. Each mql4-program should have at least one event-handler, otherwise the loaded program will not be executed. Event handlers have predefined names, parameters and return types.

Type	Function	Parameters	Application	Comment

	name			
int	OnInit	none	Expert Advisors, indicators and scripts	Init event handler. It allows to use the void return type.
void	OnDeinit	const int reason	Expert Advisors, indicators and scripts	Deinit event handler.
void	OnStart	none	scripts	Start event handler.
int	OnCalculate	const int rates_total, const int prev_calculated, const datetime &Time[], const double &Open[], const double &High[], const double &Low[], const double &Close[], const long &TickVolume[], const long &Volume[], const int &Spread[]	indicators	Calculate event handler for all prices.
void	OnTick	none	Expert Advisors	NewTick event handler. While the event of a new tick receipt is being processed, no other events of this type are received.
void	OnTimer	none	Expert Advisors and indicators	Timer event handler.
double	OnTester	none	Expert Advisors	Tester event handler.
void	OnChartEvent	const int id,	Expert	ChartEvent event handler.

		const long &lparam, const double &dparam, const string &sparam	Advisors and indicators	

A client terminal sends new events to the corresponding open charts. Events can also be generated by charts (chart events) or mql4-programs (custom events). Generation of events of creation or deletion of graphical objects on a chart can be enabled or disabled by setting CHART_EVENT_OBJECT_CREATE and CHART_EVENT_OBJECT_DELETE chart properties. Each MQL4 program and each chart has its own queue of events, where all new incoming events are added.

A program receives only events from the chart it runs on. All events are processed one after another in the order they are received. If a queue already has a NewTick event, or this event is currently being processed, then the new NewTick event is not placed in the queue of the MQL4 program. Similarly, if ChartEvent is already enqueued, or this event is being processed, no new event of this kind is enqueued. The timer events are handled the same way if the Timer event is in the queue or being handled, the new timer event is not enqueued.

Event queues have a limited but sufficient size, so that the queue overflow for well written programs is unlikely. In case of queue overflow, new events are discarded without queuing.

It is not recommended to use infinite loops to handle events. The exception to this rule may be only scripts that process only a single Start event.

Libraries do not handle any events.

Functions prohibited in Indicators and Expert Advisors

Indicators, scripts and Expert Advisors are executable programs written in MQL4. They are designed for different types of tasks. Therefore there are some restrictions on the use of certain functions, depending on the type of program. The following functions are prohibited in indicators:

· OrderSend();

· SendFTP();

· Sleep();

· ExpertRemove();

- MessageBox().

All functions designed for indicators are prohibited in Expert Advisors and scripts:

- SetIndexBuffer();
- IndicatorSetDouble();
- IndicatorSetInteger();
- IndicatorSetString().

The library is not an independent program and is executed in the context of the MQL4 program that has called it: script, indicator or Expert Advisor. Accordingly, the above restrictions apply to the called library.

Loading and Unloading of Indicators

Indicators are loaded in the following cases:

- an indicator is attached to a chart;
- terminal start (if the indicator was attached to the chart prior to the shutdown of the terminal);
- loading of a template (if the indicator attached to a chart is specified in the template);
- change of a profile (if the indicator is attached to one of the profile charts);
- change of a symbol and/or timeframe of a chart, to which the indicator is attached;
- after the successful recompilation of an indicator (if the indicator was attached to a chart);
- change of input parameters of the indicator.

Indicators are unloaded in the following cases:

- when detaching an indicator from a chart;
- terminal shutdown (if the indicator was attached to a chart);
- loading of a template (if an indicator is attached to a chart);
- closing of a chart, to which the indicator was attached;
- change of a profile (if the indicator is attached to one of charts of the changed profile);
- change of a symbol and/or timeframe of a chart, to which the indicator is attached;
- change of input parameters of the indicator.

Loading and Unloading of Expert Advisors

Expert Advisors are loaded in the following cases:

- when attaching an Expert Advisor to a chart;
- terminal start (if the Expert Advisor was attached to the chart prior to the shutdown of the terminal);
- loading of a template (if the Expert Advisor attached to the chart is specified in the template);
- change of a profile (if the Expert Advisor is attached to the one of the profile charts);
- connection to an account, even if the account number is the same (if the Expert Advisor was attached to the chart before the authorization of the terminal on the server).

Expert Advisors are unloaded in the following cases:

- when detaching an Expert Advisor from a chart;
- if a new Expert Advisor is attached to a chart, if another Expert Advisor has been attached already, this Expert Advisor is unloaded.
- terminal shutdown (if the Expert Advisor was attached to a chart);
- loading of a template (if an Expert Advisor is attached to the chart);
- close of a chart, to which the Expert Advisor is attached.
- change of a profile (if the Expert Advisor is attached to one of charts of the changed profile);
- change of the account to which the terminal is connected (if the Expert Advisor was attached to the chart before the authorization of the terminal on the server);
- calling the ExpertRemove() function.

In case the symbol or timeframe of a chart, to which the Expert Advisor is attached, changes, Expert Advisors are not loaded or unloaded. In this case client terminal subsequently calls OnDeinit() handlers on the old symbol/timeframe and OnInit() on the new symbol/timeframe (if they are such), values of global variables and static variables are not reset. All events, which have been received for the Expert Advisor before the initialization is completed (OnInit() function) are skipped.

For a better understanding of the Expert Advisor operation we recommend to compile the code of the following Expert Advisor and perform actions of load/unload, template change, symbol change, timeframe change etc:

Example:

```
//+------------------------------------------------------------------+ //|
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

class CTestClass
  {
public:
   CTestClass() { Print("CTestClass constructor"); }
   ~CTestClass() { Print("CTestClass destructor"); }
  };
CTestClass global;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   Print("Initialization");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print("Deinitialization with reason",reason);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
```

Loading and Unloading of Scripts

Scripts are loaded immediately after they are attached to a chart and unloaded immediately after they complete their operation.

When a program is unloaded (deleted from a chart) the client terminal performs deinitialization of global variables and deletes the events queue. In this case deinitialization means reset of all the string-type variables, deallocation of dynamical array objects and call of their destructors if they are available.

```
//+------------------------------------------------------------------+
//|                                                   TestScript.mq5 |
//|                        Copyright 2014, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

class CTestClass
  {
public:
   CTestClass() { Print("CTestClass constructor"); }
   ~CTestClass() { Print("CTestClass destructor"); }
  };
CTestClass global;
//+------------------------------------------------------------------+
//| Script program initialization function                           |
//+------------------------------------------------------------------+
void OnInit()
  {
   Print(__FUNCTION__);
  }
//+------------------------------------------------------------------+
//| Script program deinitialization function                         |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   Print(__FUNCTION__," reason=",reason);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   Print(__FUNCTION__);
  }
```

Input Parameters and Source Code Compilation

If a source code of a program launched on a chart is successfully recompiled, its previous version is removed from the chart and the new compiled copy is executed instead.

If the set of the input parameters is not changed after the recompilation, the previously specified parameter values are applied. Otherwise, the default values are used.

The set of mql4 program input parameters is considered to be changed when editing the source code in the following cases:

· the number of parameters has been changed;

· the sequence order of parameters has been changed;

· parameter names have been changed;

· the type of one or more parameters has been changed.

Changing a default value of any of the parameters is not considered to be a change of the input parameter set.

The set of the input parameters clearly identifies the program in the terminal's execution system. If this set is unchanged, the new versions of the executable file are considered to retain the entire logic and functionality of the program.

If the set of the input parameters is changed, the terminal considers the new executable file as incompatible with the program that has been previously launched on the chart. Thus, the new recompiled program is launched with the set of the input parameters having default values.

In other cases (including the ones when a default value of any parameter is changed), the previously specified parameters are applied after the recompilation.

The OnInit() predefined function is called after any compilation. Its purpose is to correctly initialize all global and static variables of the program. The program's input parameter values should be used correctly in OnInit() event handler.

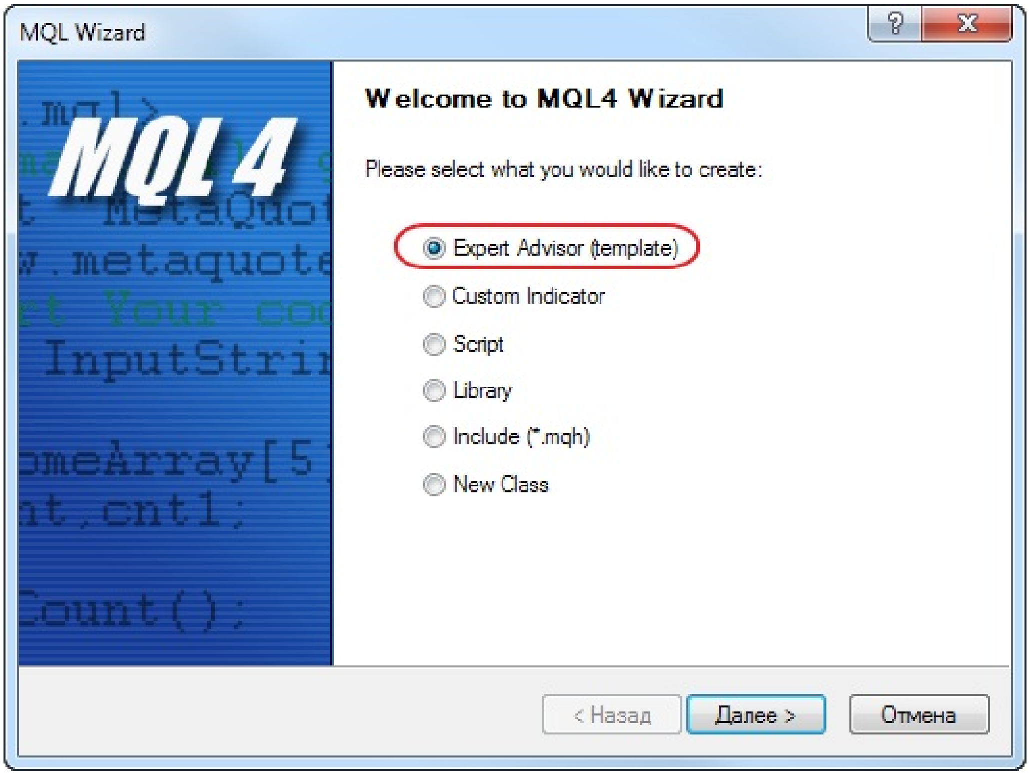
See also

Client terminal events, Event handlers

Trade Permission

Trade Automation

MQL4 language provides a special group of trade functions designed for developing automated trading systems. Programs developed for automated trading with no human intervention are called Expert Advisors or trading robots. In order to create an Expert Advisor in MetaEditor, launch MQL4 Wizard and select the option "Expert Advisor (template)". It allows you to create a template with ready-made event handling functions that should be supplemented with all necessary functionality by means of programming.



MQL Wizard

Welcome to MQL4 Wizard

Please select what you would like to create:

● Expert Advisor (template)
○ Custom Indicator
○ Script
○ Library
○ Include (*.mqh)
○ New Class

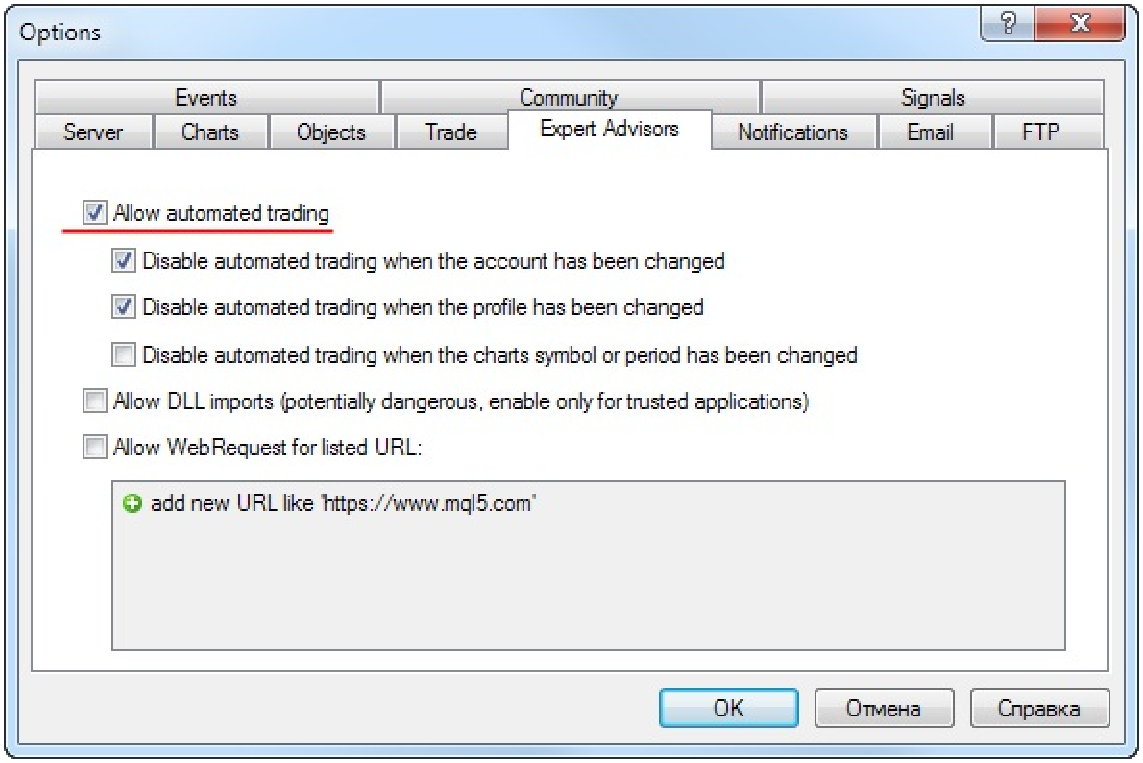
Trading functions can work only in Expert Advisors and scripts. Trading is not allowed for indicators.

Checking for Permission to Perform Automated Trading

In order to develop a reliable Expert Advisor capable of working without human intervention, it is necessary to arrange a set of important checks. First, we should programmatically check if trading is allowed at all. This is a basic check that is indispensable when developing any automated system.

Checking for permission to perform automated trading in the terminal

The terminal settings provide you with an ability to allow or forbid automated trading for all programs.



☑ Allow automated trading

☑ Disable automated trading when the account has been changed

☑ Disable automated trading when the profile has been changed

□ Disable automated trading when the charts symbol or period has been changed

□ Allow DLL imports (potentially dangerous, enable only for trusted applications)

□ Allow WebRequest for listed URL:

add new URL like 'https://www.mql5.com'

You can switch automated trading option right on the terminal's Standard panel:

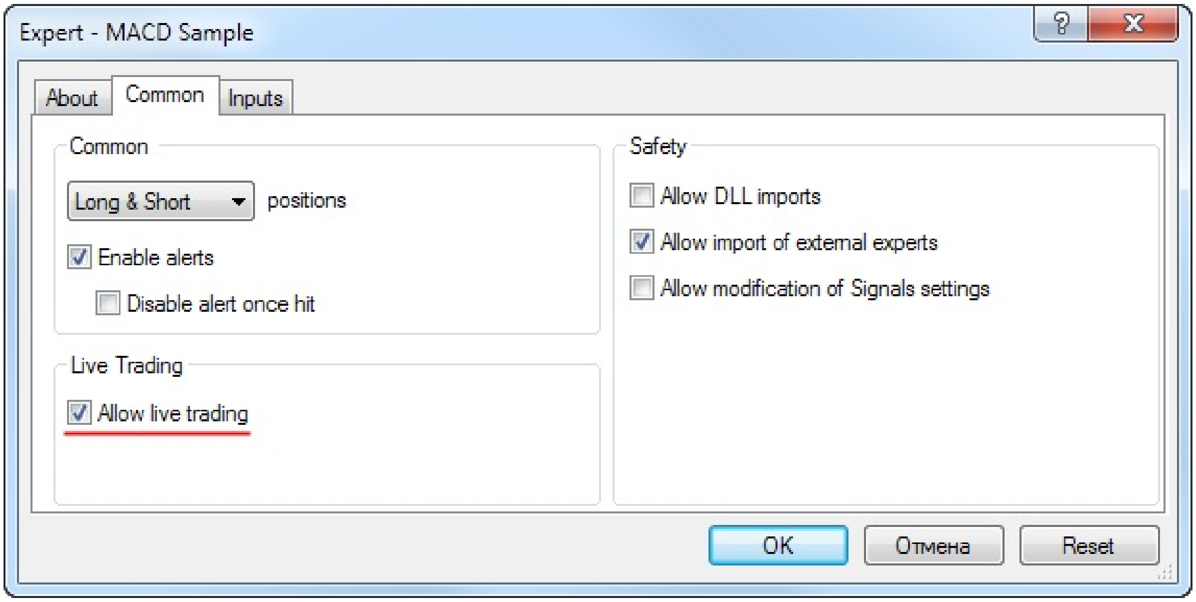
- AutoTrading automated trading enabled, trading functions in launched applications are allowed for use.
- AutoTrading automated trading disabled, running applications are unable to execute trading functions.

Sample check:

```
if (!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))     Alert("Check if auto
```

Checking if trading is allowed for a certain running Expert Advisor/script

You can allow or forbid automated trading for a certain program when launching it. To do this, use the special check box in the program properties.



Common

Long & Short positions

☑ Enable alerts

□ Disable alert once hit

Live Trading

☑ Allow live trading

Safety

□ Allow DLL imports

☑ Allow import of external experts

□ Allow modification of Signals settings

Sample check:

```
   if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
      Alert("Check if automated trading is allowed in the terminal settinç
   else
     {
      if(!MQLInfoInteger(MQL_TRADE_ALLOWED))
         Alert("Automated trading is forbidden in the program settings for
     }
```

Checking if trading is allowed for any Expert Advisors/scripts for the current account

Automated trading can be disabled at the trade server side. Sample check:

```
   if(!AccountInfoInteger(ACCOUNT_TRADE_EXPERT))
      Alert("Automated trading is forbidden for the account ",AccountInfoI
      " at the trade server side");
```

If automated trading is disabled for a trading account, trading operations of Expert Advisors/scripts are not executed.

Checking if trading is allowed for the current account

In some cases, any trading operations are disabled for a certain trading account neither manual nor automated trading can be performed. Sample check when an investor password has been used to connect to a trading account:

```
   if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
      Comment("Trading is forbidden for the account ",AccountInfoInteger(A
            ".\n Perhaps an investor password has been used to connect to
            "\n Check the terminal journal for the following entry:",
            "\n\'",AccountInfoInteger(ACCOUNT_LOGIN),"\': trading has been
```

AccountInfoInteger(ACCOUNT_TRADE_ALLOWED) may return **false** in the following cases:

- no connection to the trade server. That can be checked using TerminalInfoInteger(TERMINAL_CONNECTED));
- trading account switched to read-only mode (sent to the archive);
- trading on the account is disabled at the trade server side;
- connection to a trading account has been performed in Investor mode.

See also

Client Terminal Properties, Account Properties, Properties of a Running MQL4 Program

Client Terminal Events

Init

Immediately after the client terminal loads a program (an Expert Advisor or custom indicator) and starts the process of initialization of global variables, the Init event will be sent, which will be processed by OnInit() event handler, if there is such. This event is also generated after a financial instrument and/or chart timeframe is changed, after a program is recompiled in MetaEditor, after input parameters are changed from the setup window of an Expert Advisor or a custom indicator. An Expert Advisor is also initialized after the account is changed.

Deinit

Before global variables are deinitialized and the program (Expert Advisor or custom indicator) is unloaded, the client terminal sends the Deinit event to the program. Deinit is also generated when the client terminal is closed, when a chart is closed, right before the security and/or timeframe is changed, at a successful program re-compilation, when input parameters are changed, and when account is changed.

The deinitialization reason can be obtained from the parameter, passed to the OnDeinit() function. The OnDeinit() function run is restricted to 2.5 seconds. If during this time the function hasn't been completed, then it is forcibly terminated.

Start

The Start event is a special event for script activation after it is loaded. This event is processed by OnStart handler. The Start event is not send to Expert Advisors or custom indicators.

NewTick

The NewTick event is generated if there are new quotes, it is processed by OnTick() of Expert Advisors attached. In case when OnTick function for the previous quote is being processed when a new quote is received, the new quote will be ignored by an Expert Advisor, because the corresponding event will not enqueued.

All new quotes that are received while the program is running are ignored until the OnTick() is completed. After that the function will run only after a new quote is received. The NewTick event is generated irrespective of whether automated trade is allowed or not ("Allow/prohibit Auto trading" button). The prohibition of automated trading denotes only that sending of trade requests from an Expert Advisor is not allowed, while the Expert Advisor keeps working.

The prohibition of automated trading by pressing the appropriate button will not stop the current execution of the OnTick() function. OnTick() is not started when the window of Expert Advisor properties is open.

Calculate

The Calculate event is generated only for indicators right after the Init event is sent and at any change of price data. It is processed by the OnCalculate function.

Timer

The Timer event is periodically generated by the client terminal for the Expert Advisor that has activated the timer by the EventSetTimer function. Usually, this function is called by OnInit. Timer event processing is performed by the OnTimer function. After the operation of the Expert Advisor is completed, it is necessary to destroy the timer using the EventKillTimer function, which is usually called in the OnDeinit function.

Tester

The Tester event is generated after testing of an Expert Advisor on history data is over. The event is handled by the OnTester() function.

ChartEvent

The ChartEvent event is generated by the client terminal when a user is working with a chart:

· keystroke, when the chart window is in focus;

· graphical object created

· graphical object deleted

· mouse press on the graphical object of the chart

· move of the graphical object using the mouse

· end of text editing in LabelEdit.

Also there is a custom event ChartEvent, which can be sent to an Expert Advisor by any mql4 program by using the EventChartCustom function. The event is processed by the OnChartEvent function.

See also

Event handlers, Program running

Resources

Using graphics and sound in MQL4 programs

Programs in MQL4 allow working with sound and graphic files:

- PlaySound() plays a sound file;
- ObjectCreate() allows creating user interfaces using graphical objects OBJ_BITMAP and OBJ_BITMAP_LABEL.

PlaySound()

Example of call of the PlaySound() function:

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
void OrderSendWithAudio()
  {
      double price=Ask;
      //--- place market order to buy 1 lot
      int ticket=OrderSend(Symbol(),OP_BUY,1,price,3,0,0,"My order",16384,
      if(ticket<0)
        {
         Print("OrderSend failed with error #",GetLastError());
         //--- if error play sound from timeout.wav
         PlaySound("timeout.wav");
        }
      else
        {
         Print("OrderSend placed successfully");
         //--- if success, play sound from Ok.wav
         PlaySound("Ok.wav");
        }
  }
```

The example shows how to play sounds from files 'Ok.wav' and 'timeout.wav', which are included into the standard terminal package. These files are located in the folder **terminal_directory\Sounds**. Here **terminal_directory** is a folder, from which the MetaTrader 4 Client Terminal is started. The location of the terminal directory can be found out from an mql4 program in the following way:

```
//--- Folder, in which terminal data are stored
   string terminal_path=TerminalInfoString(TERMINAL_PATH);
```

You can use sound files not only from the folder **terminal_directory\Sounds**, but also from any subfolder located in **terminal_data_directory\MQL4.** You can find out the location of the terminal data directory from the terminal menu "File" -> "Open Data Folder" or using program method:

```
//--- Folder, in which terminal data are stored
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
```

For example, if the Demo.wav sound file is located in terminal_data_directory\MQL4\Files, then call of PlaySound() should be written the following way:

```
//--- play Demo.wav from the folder terminal_directory_data\MQL4\Files\
   PlaySound("\\Files\\Demo.wav");
```

Please note that in the comment the path to the file is written using backslash "\", and in the function "\\" is used.

When specifying the path, always use only the double backslash as a separator, because a single backslash is a control symbol for the compiler when dealing with constant strings and character constants in the program source code.

Call PlaySound() function with NULL parameter to stop playback:

```
//--- call of PlaySound() with NULL parameter stops playback
   PlaySound(NULL);
```

ObjectCreate()

Example of an Expert Advisor, which creates a graphical label (OBJ_BITMAP_LABEL) using the ObjectCreate() function.

```
string label_name="currency_label";        // name of the OBJ_BITMAP_LABEL
string euro      ="\\Images\\euro.bmp";    // path to the file terminal_da
string dollar    ="\\Images\\dollar.bmp";  // path to the file terminal_da
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create a button OBJ_BITMAP_LABEL, if it hasn't been created yet
   if(ObjectFind(0,label_name)<0)
     {
      //--- trying to create object OBJ_BITMAP_LABEL
      bool created=ObjectCreate(0,label_name,OBJ_BITMAP_LABEL,0,0,0);
      if(created)
```

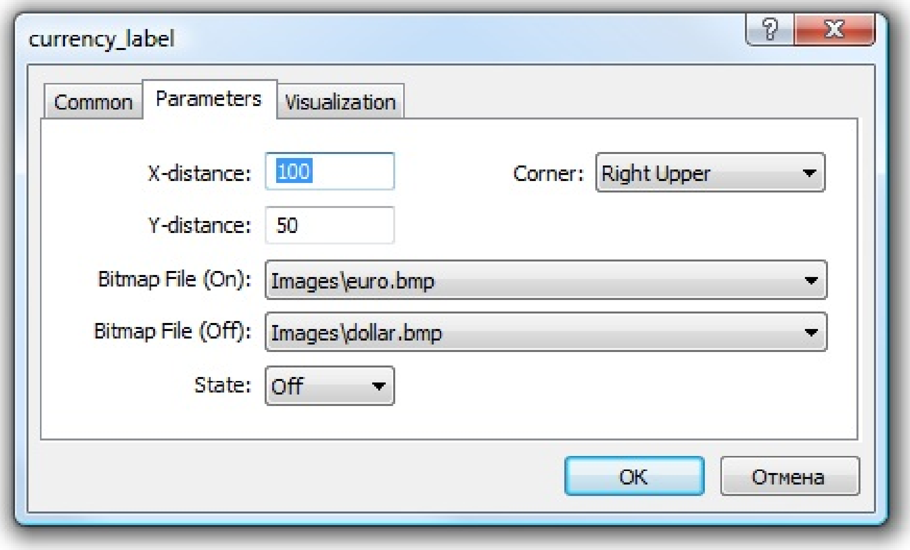
```
        {
         //--- link the button to the left upper corner of the chart
         ObjectSetInteger(0,label_name,OBJPROP_CORNER,CORNER_RIGHT_UPPER);
         //--- now set up the object properties
         ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,100);
         ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,50);
         //--- reset the code of the last error to 0
         ResetLastError();
         //--- download a picture to indicate the "Pressed" state of the b
         bool set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,0,euro);
         //--- test the result
         if(!set)
           {
            PrintFormat("Failed to download image from file %s. Error code
           }
         ResetLastError();
         //--- download a picture to indicate the "Unpressed" state of the
         set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,1,dollar);

         if(!set)
           {
            PrintFormat("Failed to download image from file %s. Error code
           }
         //--- send a command for a chart to refresh so that the button ap
         ChartRedraw(0);
        }
      else
        {
         //--- failed to create an object, notify
         PrintFormat("Failed to create object OBJ_BITMAP_LABEL. Error code
        }
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- delete an object from a chart
   ObjectDelete(0,label_name);
  }
```

Creation and setup of the graphical object named currency_label are carried out in the OnInit() function. The paths to the graphical files are set in global variables **euro** and **dollar**, a double backlash is used for a separator:

```
string euro        ="\\Images\\euro.bmp";     // path to the file terminal_da
string dollar      ="\\Images\\dollar.bmp";   // path to the file terminal_da
```

The files are located in the folder **terminal_data_directory**\MQL4\Images.

Object OBJ_BITMAP_LABEL is actually a button, which displays one of the two images, depending on the button state (pressed or unpressed): euro.bmp or dollar.bmp.



currency_label

X-distance: 100

Corner: Right Upper

Y-distance: 50

Bitmap File (On): Images\euro.bmp

Bitmap File (Off): Images\dollar.bmp

State: Off

The size of the button with a graphical interface is automatically adjusted to the size of the picture. The image is changed by a left mouse button click on the OBJ_BITMAP_LABEL object (**"Disable selection" option must be checked in the properties**). The OBJ_BITMAP object is created the same way - it is used for creating the background with a necessary image.

The value of the OBJPROP_BMPFILE property, which is responsible for the appearance of the objects OBJ_BITMAP and OBJ_BITMAP_LABEL, can be changed dynamically. This allows creating various interactive user interfaces for mql4 programs.

Including resources to executable files during compilation of mql4 programs

An mql4 program may need a lot of different downloadable resources in the form of image and sound files. In order to eliminate the need to transfer all these files when moving an executable file in MQL4, the compiler's directive #resource should be used:

```
 #resource path_to_resource_file
```

The #resource command tells the compiler that the resource at the specified path **path_to_resource_file** should be included into the executable EX5 file.

Thus all the necessary images and sounds can be located directly in an EX4 file, so that there is no need to transfer separately the files used in it, if you want to run the program on a different terminal. Any EX4 file can contain resources, and any EX4 program can use resources from another EX4 program.

The files in format BMP and WAV are automatically compressed before including them to an EX4 file. This denotes that in addition to the creation of complete programs in MQL4, using resources also allows to reduce the total size of necessary files when using graphics and sounds, as compared to the usual way of MQL4 program writing.

The resource file size must not exceed 16 Mb.

Search for specified resources by a compiler

A resource is inserted using the command #resource "<path to the resource file>"

```
#resource "<path_to_resource_file>"
```

The length of the constant string <path_to_resource_file> must not exceed 63 characters.

The file and folder names, included as resources, must be in English.

The compiler searches for a resource at the specified path in the following order:

- if the backslash "\" separator (written as "\\") is placed at the beginning of the path, it searches for the resource relative to the directory **terminal_data_directory**\MQL4\,
- if there is no backslash, it searches for the resource relative to the location of the source file, in which the resource is written.

The resource path cannot contain the substrings "..\\" and ":\\".

Examples of resource inclusion:

```
//--- correct specification of resources
#resource "\\Images\\euro.bmp" // euro.bmp is located in terminal_data_dir
#resource "picture.bmp"        // picture.bmp is located in the same direc
#resource "Resource\\map.bmp"  // the resource is located in source_file_d

//--- incorrect specification of resources
#resource ":picture_2.bmp"     // must not contain ":"
#resource "..\\picture_3.bmp"  // must not contain ".."
#resource "\\Files\\Images\\Folder_First\\My_panel\\Labels\\too_long_path.
```

Use of Resources

Resource name

After a resource is declared using the #resource directive, it can be used in any part of a program. The name of the resource is its path without a backslash at the beginning of the line, which sets the path to the resource. To use your own resource in the code, the special sign "::" should be added before the resource name.

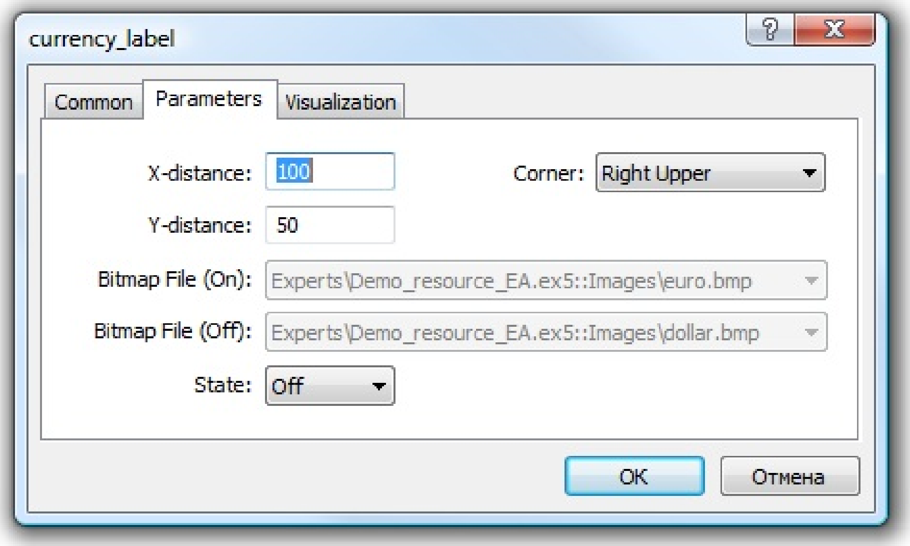
Examples:

```
//--- examples of resource specification and their names in comments
#resource "\\Images\\euro.bmp"          // resource name - Images\euro.bmp
#resource "picture.bmp"                 // resource name - picture.bmp
#resource "Resource\\map.bmp"           // resource name - Resource\map.bm
#resource "\\Files\\Pictures\\good.bmp" // resource name - Files\Pictures\
#resource "\\Files\\Demo.wav";          // resource name - Files\Demo.wav"
#resource "\\Sounds\\thrill.wav";       // resource name - Sounds\thrill.w
...

//--- utilization of resources
ObjectSetString(0,bitmap_name,OBJPROP_BMPFILE,0,"::Images\\euro.bmp");
...
ObjectSetString(0,my_bitmap,OBJPROP_BMPFILE,0,"::picture.bmp");
...
set=ObjectSetString(0,bitmap_label,OBJPROP_BMPFILE,1,"::Files\\Pictures\\g
...
PlaySound("::Files\\Demo.wav");
...
PlaySound("::Sounds\\thrill.wav");
```

It should be noted that when setting images from a resource to the OBJ_BITMAP and OBJ_BITMAP_LABEL objects, the value of the OBJPROP_BMPFILE property cannot be modified manually. For example, for creating OBJ_BITMAP_LABEL we use resources euro.bmp and dollar.bmp.

```
#resource "\\Images\\euro.bmp";    // euro.bmp is located in terminal_data
#resource "\\Images\\dollar.bmp";  // dollar.bmp is located in terminal_da
```

When viewing the properties of this object, we'll see that the properties BitMap File (On) and BitMap File (Off) are dimmed and cannot be change manually:



X-distance: 100
Corner: Right Upper
Y-distance: 50
Bitmap File (On): Experts\Demo_resource_EA.ex5::Images\euro.bmp
Bitmap File (Off): Experts\Demo_resource_EA.ex5::Images\dollar.bmp
State: Off

Using the resources of other mql4 programs

There is another advantage of using resources in any MQL4 program, resources of another EX4 file can be used. Thus the resources from one EX4 file can be used in many other mql4 programs.

In order to use a resource name from another file, it should be specified as <path_EX4_file_name>::<resource_name>. For example, suppose the Draw_Triangles_Script.mq5 script contains a resource to an image in the file triangle.bmp:

```
 #resource "\\Files\\triangle.bmp"
```

Then its name, for using in the script itself, will look like "Files\triangle.bmp", and in order to use it, "::" should be added to the resource name.

```
//--- using the resource in the script
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"::Files\\triangle.bmp"
```

In order to use the same resource from another program, e.g. from an Expert Advisor, we need to add to the resource name the path to the EX4 file relative to **terminal_data_directory\MQL4\** and the name of the script's EX4 file - **Draw_Triangles_Script.ex4**. Suppose the script is located in the standard folder **terminal_data_directory\MQL4\**Scripts\, then the call should be written the following way:

```
//--- using a resource from a script in an EA
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"\\Scripts\\Draw_Triang
```

If the path to the executable file is not specified when calling the resource from another EX4, the executable file is searched for in the same folder that contains the program that calls the resource. This means that if an Expert

Advisor calls a resource from Draw_Triangles_Script.ex4 without specification of the path, like this:

```
//--- call script resource in an EA without specifying the path
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"Draw_Triangles_Script.
```

then the file will be searched for in the folder **terminal_data_directory\MQL4**\Experts\, if the Expert Advisor is located in **terminal_data_directory\MQL4**\Experts\.

Working with custom indicators included as resources

One or several custom indicators may be necessary for the operation of MQL4 applications. All of them can be included into the code of an executable MQL5 program. Inclusion of indicators as resources simplifies the distribution of applications.

Below is an example of including and using SampleIndicator.ex4 custom indicator located in **terminal_data_folder**\MQL4\Indicators\ directory:

```
//+------------------------------------------------------------------+
//|                                                      SampleEA.mq4 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#resource "\\Indicators\\SampleIndicator.ex4"
#property strict
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- get custom indicator value
   double value=iCustom(_Symbol,_Period,"::Indicators\\SampleIndicator.ex4
   PrintFormat("Indicator: iCustom value=%f",value);
//--- ...
   return(INIT_SUCCEEDED);
  }
```

The case when a custom indicator in OnInit() function creates one or more copies of itself requires special consideration. Please keep in mind that the resource should be specified in the following way: **<path_EX4_file_name>::<resource_name>**.

For example, if SampleIndicator.ex4 indicator is included to SampleEA.ex4 Expert Advisor as a resource, the path to itself specified when calling iCustom() in the custom indicator's initialization function looks the following way: "\\Experts\\SampleEA.ex4::Indicators\\SampleIndicator.ex4". When this path is set explicitly, SampleIndicator.ex4 custom indicator is rigidly connected to SampleEA.ex4 Expert Advisor losing ability to work independently.

The path to itself can be received using GetRelativeProgramPath() function. The example of its usage is provided below:

```
//+------------------------------------------------------------------+
//|                                              SampleIndicator.mq4 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property indicator_separate_window
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- the wrong way to provide a link to itself
//--- string path="\\Experts\\SampleEA.ex4::Indicators\\SampleIndicator.ex
//--- the right way to receive a link to itself
   string path=GetRelativeProgramPath();
//--- get indicator value
   double value=iCustom(_Symbol,_Period,path,0,0);
   PrintFormat("Path=%s, iCustom value=%f",path,value);
//---
   return(INIT_SUCCEEDED);
  }
///....
//+------------------------------------------------------------------+
//| GetRelativeProgramPath                                           |
//+------------------------------------------------------------------+
string GetRelativeProgramPath()
  {
   int pos2;
//--- get the absolute path to the application
   string path=MQLInfoString(MQL_PROGRAM_PATH);
//--- find the position of "\MQL4\" substring
   int    pos =StringFind(path,"\\MQL4\\");
//--- substring not found - error
   if(pos<0)
      return(NULL);
//--- skip "\MQL4" directory
   pos+=5;
//--- skip extra '\' symbols
```

```
   while(StringGetCharacter(path,pos+1)=='\\')
      pos++;
//--- if this is a resource, return the path relative to MQL5 directory
   if(StringFind(path,"::",pos)>=0)
      return(StringSubstr(path,pos));
//--- find a separator for the first MQL4 subdirectory (for example, MQL4\
//--- if not found, return the path relative to MQL4 directory
   if((pos2=StringFind(path,"\\",pos+1))<0)
      return(StringSubstr(path,pos));
//--- return the path relative to the subdirectory (for example, MQL4\Indi
   return(StringSubstr(path,pos2+1));
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double& price[])
  {
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

ResourceCreate(), ResourceSave(), PlaySound(), ObjectSetInteger(), ChartApplyTemplate(), File Functions

Call of Imported Functions

To import functions during the execution of a mql4-program, the client terminal uses early binding. This means that if a program has call of an imported function, the corresponding module (ex4 or dll) is loaded during the program load. MQL4 and DLL libraries are executed in the thread of a calling module.

It is not recommended to use the fully specified name of the module to be loaded like *Drive:\Directory\FileName.Ext*. The MQL4 libraries are loaded from the *terminal_dir\MQL4\Libraries* folder. If the library hasn't been found, then the client terminal performs an attempt to load it from *terminal_dir\experts* folder.

The system libraries (DLL) are loaded by the operating system rules. If the library is already loaded (for example, another Expert Advisor, and even from another client terminal, running in parallel), then it uses requests to the library already loaded. Otherwise, it performs a search in the following sequence:

1. Directory, from which the module importing dll was started. The module here is an Expert Advisor, a script, an indicator or EX4 library;
2. Directory terminal_data_directory\MQL4\Libraries (TERMINAL_DATA_PATH\MQL4\Libraries);
3. Directory, from which the MetaTrader 4 client terminal was started;
4. System directory;
5. Windows directory;
6. Current directory;
7. Directories listed in the PATH system variable.

If the DLL library uses another DLL in its work, the first one cannot be loaded in case when there is no second DLL.

Before an Expert Advisor (script, indicator) is loaded, a common list of all EX4 library modules is formed. It's going to be used both from a loaded Expert Advisor (script, indicator) and from libraries of this list. Thus the one-time loading of many times used EX4 library modules is needed. Libraries use predefined variables of the Expert Advisor (script, indicator) they were called by.

The imported library EX4 is searched for in the following sequence:

1. Directory, path to which is set relative to the directory of the Expert Advisor

(script, indicator) that imports EX4;
2. Directory terminal_directory\MQL4\Libraries;
3. Directory MQL4\Libraries in the common directory of all MetaTrader 4 client terminals (Common\MQL4\Libraries).

Functions imported DLL into a mql4-program must ensure the Windows API calls agreement. To ensure such an agreement, in the source text of programs written in C or C++, use the keyword __stdcall, which is specific to the Microsoft(r) compilers. This agreement is characterized by the following:

- caller (in our case it is a mql4-program) should "see" a prototype of a function called (imported from the DLL), in order to properly combine parameters to a stack;
- caller (in our case it is a mql4-program) puts parameters to the stack in a reverse order, from right to left - in this order an imported function reads parameters passed to it;
- parameters are passed by value, except those explicitly passed by reference (in our case strings)
- an imported function cleans the stack independently by reading parameters passed to it.

When describing the prototype of an imported function, default parameters can be used.

If the corresponding library is unable to load, or there is a prohibition on the DLL use, or the imported function is not found - the Expert Advisor stops its operation with the appropriate message "Expert Advisor stopped" in the Journal (log file). In this case the Expert Advisor will not run until it is reinitialized. An Expert Advisor can be reinitialized as a result of recompilation or after the table of its properties is opened and OK is pressed.

Passing Parameters

All parameters of simple types are passed by values unless it is explicitly indicated that they are passed by reference. When a string is passed, the address of the buffer of the copied string is passed; if a string is passed by reference, the address of the buffer of this string without copying it is passed to the function imported from DLL.

Structures that contain dynamic arrays, strings, classes, other complex structures, as well as static or dynamic arrays of the enumerated objects, can't be passed as a parameter to an imported function.

When passing an array to DLL, the address of the beginning of the data buffer is always passed (irrespective of the AS_SERIES flag). A function inside a DLL knows nothing about the AS_SERIES flag, the passed array is a static array of an undefined length; an additional parameter should be used for specifying the array size.

Runtime Errors

The executing subsystem of the client terminal has an opportunity to save the error code in case it occurs during a MQL4 program run. There is a predefined variable _LastError for each executable MQL4 program.

Before starting the OnInit function, the _LastError variable is reset. In case an erroneous situation occurs during calculations or in the process of internal function calls, the _LastError variable accepts a corresponding error code. The value stored in this variable can be obtained using the GetLastError() function.

There are several critical errors in case of which a program is terminated immediately:

- division by zero
- going beyond array boundary
- using an incorrect object pointer

Operation of Programs in the Strategy Tester

The Strategy Tester in MetaTrader 4 trading terminal allows you to test Expert Advisor's performance on historical data.

The features of program testing and optimization in the Strategy Tester should be considered when testing a trading robot:

- Function limitations in the Strategy Tester
- The global variables of the client terminal
- Simulation of time in the Strategy Tester
- Graphical objects in testing
- Event handling in the tester

Function Limitations in the Strategy Tester

There are operation limitations for some functions in the client terminal's Strategy Tester. Calling that functions leads to error 4059 (Function is not allowed in testing mode).

The Sleep() Function

The Sleep() function does not cause any delays in the Strategy Tester.

The Print() and PrintFormat() Functions

To increase performance, Print() and PrintFormat() functions are not executed when optimizing the trading robot's parameters. The exception is the use of these functions inside the OnInit() handler. This allows you to easily find the cause of errors when they occur.

The Alert(), MessageBox(), PlaySound(), SendFTP, SendMail(), SendNotification(), WebRequest() Functions

The Alert(), MessageBox(), PlaySound(), SendFTP(), SendMail(), SendNotification() и WebRequest() functions designed for interaction with the "outside world" are not executed in the Strategy Tester.

The OrderSend(), OrderModify(), OrderDelete(), OrderClose(), OrderCloseBy() Functions

Trade operations are not performed on the symbols that are different from the tested one.

The Global Variables of the Client Terminal

Since the tester is the part of the client terminal, they share the common global variables. Thus, their names should not overlap with the names of the global variables of working applications. This may lead to incorrect operation of programs and inaccurate test results.

Simulation of Time in the Tester

When testing, the time is simulated according to the historical data. TimeLocal() local time is always equal to TimeCurrent() server time. In turn, the server time is always equal to the time corresponding to the GMT - TimeGMT(). This way, all of these functions display the same time during testing.

The absence of the difference between GMT, local and server time in the tester is provided deliberately in case connection to the server is lost. The test results should always be the same, regardless of whether or not there is a connection. Information about the server time is not stored locally, and is taken from the server.

Graphical Objects in Testing

During visualization, the Expert Advisor interacts with a real chart. In case there is no visualization, the Expert Advisor works with a "virtual" chart that is not displayed. The former case has some peculiarities. During optimization, working with graphical objects is not supported.

Event Handling in the Tester

The following events are handled in the Strategy Tester: initializing an Expert Advisor before a single run of OnInit(), deinitializing an Expert Advisor after a single run of OnDeInit() and simulating a new tick OnTick().

In addition, Tester event handled in OnTester() function is generated before calling OnDeInit() deinitialization function after testing a trading robot on historical data. The value returned by this function is used as a Custom max criterion when optimizing the input parameters.

Timer and ChartEvent events are not handled in the Strategy Tester.

The predefined Variables

For each executable mql4-program a set of predefined variables is supported, which reflect the state of the current price chart by the moment a mql4-program (Expert Advisor, script or custom indicator) is started.

Values of predefined variables are set by the client terminal before a mql4-program is started. Predefined variables are constant and cannot be changed from a mql4-program. As exception, there is a special variable _LastError, which can be reset to 0 by the ResetLastError function.

Variable	Value
_Digits	Number of decimal places
_Point	Size of the current symbol point in the quote currency
_LastError	The last error code
_Period	Timeframe of the current chart
_RandomSeed	Current status of the generator of pseudo-random integers
_StopFlag	Program stop flag
_Symbol	Symbol name of the current chart
_UninitReason	Uninitialization reason code
Ask	The latest known seller's price (ask price) of the current symbol
Bars	Number of bars in the current chart
Bid	The latest known buyer's price (offer price, bid price) of the current symbol
Close	Series array that contains close prices for each bar of the current chart
Digits	Number of digits after decimal point for the current symbol prices
High	Series array that contains the highest prices of each bar of the current chart
Low	Series array that contains the lowest prices of each bar of the current chart
Open	Series array that contains open prices of each bar of the current chart
Point	The current symbol point value in the quote currency
Time	Series array that contains open time of each bar of the current chart
Volume	Series array that contains tick volumes of each bar of the current

	chart

Predefined variables cannot be defined in a library. A library uses such variables that are defined in program from which this library is called.

Example:

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
void OnStart()
  {
   Print("Symbol name of the current chart=",_Symbol);
   Print("Timeframe of the current chart=",_Period);
   Print("The latest known seller's price (ask price) for the current symb
   Print("The latest known buyer's price (bid price) of the current symbol
   Print("Number of decimal places=",Digits);
   Print("Number of decimal places=",_Digits);
   Print("Size of the current symbol point in the quote currency=",_Point)
   Print("Size of the current symbol point in the quote currency=",Point);
   Print("Number of bars in the current chart=",Bars);
   Print("Open price of the current bar of the current chart=",Open[0]);
   Print("Close price of the current bar of the current chart=",Close[0]);
   Print("High price of the current bar of the current chart=",High[0]);
   Print("Low price of the current bar of the current chart=",Low[0]);
   Print("Time of the current bar of the current chart=",Time[0]);
   Print("Tick volume of the current bar of the current chart=",Volume[0])
   Print("Last error code=",_LastError);
   Print("Random seed=",_RandomSeed);
   Print("Stop flag=",_StopFlag);
   Print("Uninitialization reason code=",_UninitReason);
  }
```

int _Digits

The _Digits variable stores number of digits after a decimal point, which defines the price accuracy of the symbol of the current chart.

You may also use the Digits() function.

double _Point

The _Point variable contains the point size of the current symbol in the quote currency.

You may also use the Point() function.

int _LastError

The _LastError variable contains code of the last error, that occurred during the mql4-program run. Its value can be reset to zero by ResetLastError().

To obtain the code of the last error, you may also use the GetLastError() function.

int _Period

The _Period variable contains the value of the timeframe of the current chart.

Also you may use the Period() function.

See also

PeriodSeconds(), Chart timeframes, Date and Time, Visibility of objects

_RandomSeed

Variable for storing the current state when generating pseudo-random integers. _RandomSeed changes its value when calling MathRand(). Use MathSrand() to set the required initial condition.

x random number received by MathRand() function is calculated in the following way at each call:

```
x=_RandomSeed*214013+2531011; _RandomSeed=x;
x=(x>>16)&0x7FFF;
```

See also

MathRand(), MathSrand(), Integer types

bool _StopFlag

The _StopFlag variable contains the flag of the mql4-program stop. When the client terminal is trying to stop the program, it sets the _StopFlag variable to true.

To check the state of the _StopFlag you may also use the IsStopped() function.

string _Symbol

The _Symbol variable contains the symbol name of the current chart.

You may also use the Symbol() function.

int _UninitReason

The _UninitReason variable contains the code of the program uninitialization reason.

Usually, this code is obtained by UninitializeReason()the function.

double Ask

The latest known seller's price (ask price) for the current symbol. The RefreshRates() function must be used to update.

Example:

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

See also

MarketInfo()

int Bars

Number of bars in the current chart.

Example:

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

See also

Function Bars, iBars

double Bid

The latest known buyer's price (offer price, bid price) of the current symbol. The RefreshRates() function must be used to update.

Example:

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

See also

MarketInfo()

double Close[]

Series array that contains close prices for each bar of the current chart.

Series array elements are indexed in the reverse order, i.e., from the last one to the first one. The current bar which is the last in the array is indexed as 0. The oldest bar, the first in the chart, is indexed as Bars-1.

Example:

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

See also

iClose()

int Digits

Number of digits after decimal point for the current symbol prices.

Example:

```
  Print(DoubleToStr(Close[0], Digits));
```

See also

MarketInfo()

double High[]

Series array that contains the highest prices of each bar of the current chart.

Series array elements are indexed in the reverse order, i.e., from the last one to the first one. The current bar which is the last in the array is indexed as 0. The oldest bar, the first in the chart, is indexed as Bars-1.

Example:

```
//---- 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--;
     }
//----
```

See also

iHigh()

double Low[]

Series array that contains the lowest prices of each bar of the current chart.

Series array elements are indexed in the reverse order, i.e., from the last one to the first one. The current bar which is the last in the array is indexed as 0. The oldest bar, the first in the chart, is indexed as Bars-1.

Example:

```
//---- minima 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--;
     }
//----
```

See also

iLow()

double Open[]

Series array that contains open prices of each bar of the current chart.

Series array elements are indexed in the reverse order, i.e., from the last one to the first one. The current bar which is the last in the array is indexed as 0. The oldest bar, the first in the chart, is indexed as Bars-1.

Example:

```
   i = Bars - counted_bars - 1;    while(i>=0)
     {
      double high  = High[i];
      double low   = Low[i];
      double open  = Open[i];
      double close = Close[i];
      AccumulationBuffer[i] = (close-low) - (high-close);
      if(AccumulationBuffer[i] != 0)
        {
         double diff = high - low;
         if(0==diff)
            AccumulationBuffer[i] = 0;
         else
           {
            AccumulationBuffer[i] /= diff;
            AccumulationBuffer[i] *= Volume[i];
           }
        }
      if(i<Bars-1) AccumulationBuffer[i] += AccumulationBuffer[i+1];
      i--;
     }
```

See also

iOpen()

double Point

The current symbol point value in the quote currency.

Example:

```
  OrderSend(Symbol(),OP_BUY,Lots,Ask,3,0,NormalizeDouble(Ask+TakeProfit*Pc
```

See also

MarketInfo()

datetime Time[]

Series array that contains open time of each bar of the current chart. Data like datetime represent time, in seconds, that has passed since 00:00 a.m. of 1 January, 1970.

Series array elements are indexed in the reverse order, i.e., from the last one to the first one. The current bar which is the last in the array is indexed as 0. The oldest bar, the first in the chart, is indexed as Bars-1.

Example:

```
for(i=Bars-2; i>=0; i--)      {
   if(High[i+1] > LastHigh) LastHigh = High[i+1];
   if(Low[i+1] < LastLow)   LastLow  = Low[i+1];
   //----
   if(TimeDay(Time[i]) != TimeDay(Time[i+1]))
     {
      P = (LastHigh + LastLow + Close[i+1])/3;
      R1 = P*2 - LastLow;
      S1 = P*2 - LastHigh;
      R2 = P + LastHigh - LastLow;
      S2 = P - (LastHigh - LastLow);
      R3 = P*2 + LastHigh - LastLow*2;
      S3 = P*2 - (LastHigh*2 - LastLow);
      LastLow  = Open[i];
      LastHigh = Open[i];
     }
   //----
   PBuffer[i]  = P;
   S1Buffer[i] = S1;
   R1Buffer[i] = R1;
   S2Buffer[i] = S2;
   R2Buffer[i] = R2;
   S3Buffer[i] = S3;
   R3Buffer[i] = R3;
  }
```

See also

iTime()

long Volume[]

Series array that contains tick volumes of each bar of the current chart.

Series array elements are indexed in the reverse order, i.e., from the last one to the first one. The current bar which is the last in the array is indexed as 0. The oldest bar, the first in the chart, is indexed as Bars-1.

Example:

```
      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);
```

See also

iVolume()

Common Functions

General-purpose functions not included into any specialized group are listed here.

Function	Action
Alert	Displays a message in a separate window
CheckPointer	Returns the type of the object pointer
Comment	Outputs a comment in the left top corner of the chart
CryptEncode	Transforms the data from array with the specified method
CryptDecode	Performs the inverse transformation of the data from array
DebugBreak	Program breakpoint in debugging
ExpertRemove	Stops Expert Advisor and unloads it from the chart
GetPointer	Returns the object pointer
GetTickCount	Returns the number of milliseconds that have elapsed since the system was started
GetMicrosecondCount	Returns the number of microseconds that have elapsed since the start of MQL4-program
MessageBox	Creates, displays a message box and manages it
PeriodSeconds	Returns the number of seconds in the period
PlaySound	Plays a sound file
Print	Displays a message in the log
PrintFormat	Formats and prints the sets of symbols and values in a log file in accordance with a preset format
ResetLastError	Sets the value of a predetermined variable _LastError to zero
ResourceCreate	Creates an image resource based on a data set
ResourceFree	Deletes dynamically created resource (freeing the memory allocated for it)
ResourceReadImage	Reads data from the graphical resource created by ResourceCreate() function or saved in EX4 file during compilation
ResourceSave	Saves a resource into the specified file
SendFTP	Sends a file at the address specified in the settings window of the "FTP" tab

SendMail	Sends an email at the address specified in the settings window of the "Email" tab
SendNotification	Sends push notifications to mobile terminals, whose MetaQuotes ID are specified in the "Notifications" tab
Sleep	Suspends execution of the current Expert Advisor or script within a specified interval
TerminalClose	Commands the terminal to complete operation
TesterStatistics	It returns the value of a specified statistic calculated based on testing results
WebRequest	Sends HTTP request to the specified server
ZeroMemory	Resets a variable passed to it by reference. The variable can be of any type, except for classes and structures that have constructors.

Alert

Displays a message in a separate window.

```
void  Alert(    argument,     // first value
   ...            // other values
   );
```

Parameters

argument

[in] Any values separated by commas. To split the information output in several lines you can use the line feed characters "\r\n". The number of parameters can not exceed 64.

Return Value

No return value.

Note

Arrays can't be passed to the Alert() function. Arrays should be output elementwise. Data of the double type are output with 8 digits after the decimal point, data of the float type are displayed with 5 digits after the decimal point. To output the real numbers with a different precision or in a scientific format, use the DoubleToString() function.

Data of the bool type is output as "true" or "false" strings. Dates are output as YYYY.MM.DD HH:MI:SS. To display a date in another format use the TimeToString() function. Data of the color type are output either as an R,G,B string or as a color name, if the color is present in a color set.

Alert() function does not work in the Strategy Tester.

See also

Comment(), Print()

CheckPointer

The function returns the type of the object pointer.

```
ENUM_POINTER_TYPE  CheckPointer(    object* anyobject      // object pointe
   );
```

Parameters

anyobject

[in] Object pointer.

Return value

Returns a value from the ENUM_POINTER_TYPE enumeration.

Note

An attempt to call an incorrect pointer results in the critical termination of a program. That's why it's necessary to call the CheckPointer function before using a pointer. A pointer can be incorrect in the following cases:

· the pointer is equal to NULL;

· the object has been deleted using the delete operator.

This function can be used for checking pointer validity. A non-zero value warranties that the pointer can be used for accessing.

Example:

```
//+------------------------------------------------------------------+
//| Deletes list by deleting its elements                            |
//+------------------------------------------------------------------+
void CMyList::Destroy()
  {
//--- service pointer for working in the loop
   CItem* item;
//--- go through loop and try to delete dynamic pointers
   while(CheckPointer(m_items)!=POINTER_INVALID)
     {
      item=m_items;
      m_items=m_items.Next();
      if(CheckPointer(item)==POINTER_DYNAMIC)
        {
         Print("Dynamyc object ",item.Identifier()," to be deleted");
         delete (item);
        }
      else Print("Non-dynamic object ",item.Identifier()," cannot be delet
     }
//---
  }
```

See also

Object Pointers, Checking the Object Pointer, Object Delete Operator delete

Comment

This function outputs a comment defined by a user in the top left corner of a chart.

```
void  Comment(    argument,      // first value
   ...           // next values
   );
```

Parameters

...

[in] Any values, separated by commas. To delimit output information into several lines, a line break symbol "\n" or "\r\n" is used. Number of parameters cannot exceed 64. Total length of the input comment (including invisible symbols) cannot exceed 2045 characters (excess symbols will be cut out during output).

Return Value

No return value.

Note

Arrays can't be passed to the Comment() function. Arrays must be entered element-by-element.

Data of double type are output with the accuracy of up to 16 digits after a decimal point, and can be output either in traditional or in scientific format, depending on what notation will be more compact. Data of float type are output with 5 digits after a decimal point. To output real numbers with another accuracy or in a predefined format, use the DoubleToString() function.

Data of bool type are output as "true" or "false" strings. Dates are shown as YYYY.MM.DD HH:MI:SS. To show dates in another format, use the TimeToString() function. Data of color type are output either as R,G,B string or as a color name, if this color is present in the color set.

Example:

```
void OnTick()
  {
//---
   double Ask,Bid;
   int Spread;
   Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
//--- Output values in three lines
   Comment(StringFormat("Show prices\nAsk = %G\nBid = %G\nSpread = %d",Ask
  }
```

See also

Alert(), Print()

CryptEncode

Transforms the data from array with the specified method.

```
int  CryptEncode(   ENUM_CRYPT_METHOD   method,        // method
   const uchar&        data[],        // source array
   const uchar&        key[],         // key
   uchar&             result[]       // destination array
   );
```

Parameters

method

[in] Data transformation method. Can be one of the values of ENUM_CRYPT_METHOD enumeration.

data[]

[in] Source array.

key[]

[in] Key array.

result[]

[out] Destination array.

Returned value

Amount of bytes in the destination array or 0 in case of error. To obtain information about the error call the GetLastError() function.

Example:

```
//+------------------------------------------------------------------+
//| ArrayToHex                                                       |
//+------------------------------------------------------------------+
string ArrayToHex(uchar &arr[],int count=-1)
  {
   string res="";
//--- check
   if(count<0 || count>ArraySize(arr))
      count=ArraySize(arr);
//--- transform to HEX string
   for(int i=0; i<count; i++)
      res+=StringFormat("%.2X",arr[i]);
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string text="The quick brown fox jumps over the lazy dog";
   string keystr="ABCDEFG";
   uchar src[],dst[],key[];
//--- prepare key
   StringToCharArray(keystr,key);
//--- copy text to source array src[]
   StringToCharArray(text,src);
//--- print initial data
   PrintFormat("Initial data: size=%d, string='%s'",ArraySize(src),CharArr
//--- encrypt src[] with DES 56-bit key in key[]
   int res=CryptEncode(CRYPT_DES,src,key,dst);
//--- check error
   if(res>0)
     {
      //--- print encrypted data
      PrintFormat("Encoded data: size=%d %s",res,ArrayToHex(dst));
      //--- decode dst[] to src[]
      res=CryptDecode(CRYPT_DES,dst,key,src);
      //--- check error
      if(res>0)
        {
         //--- print decoded data
         PrintFormat("Decoded data: size=%d, string='%s'",ArraySize(src),C
        }
      else
         Print("Error in CryptDecode. Error code=",GetLastError());
     }
   else
      Print("Error in CryptEncode. Error code=",GetLastError());
  }
```

See also

Array Functions, CryptDecode()

CryptDecode

Performs the inverse transformation of the data from array, tranformed by CryptEncode().

```
int  CryptEncode(    ENUM_CRYPT_METHOD    method,          // method
   const uchar&          data[],         // source array
   const uchar&          key[],          // key
   uchar&                result[]        // destination array
   );
```

Parameters

method

[in] Data transformation method. Can be one of the values of ENUM_CRYPT_METHOD enumeration.

data[]

[in] Source array.

key[]

[in] Key array.

result[]

[out] Destination array.

Returned value

Amount of bytes in the destination array or 0 in case of error. To obtain information about the error call the GetLastError() function.

See also

Array Functions, CryptEncode()

DebugBreak

It is a program breakpoint in debugging.

```
void  DebugBreak();
```

Return Value

No return value.

Note

Execution of an MQL4 program is interrupted only if a program is started in a debugging mode. The function can be used for viewing values of variables and/or for further step-by-step execution.

ExpertRemove

The function stops an Expert Advisor and unloads it from a chart.

```
void  ExpertRemove();
```

Returned value

No return value.

Note

The Expert Advisor is not stopped immediately as you call ExpertRemove(); just a flag to stop the EA operation is set. That is, any next event won't be processed, OnDeinit() will be called and the Expert Advisor will be unloaded and removed from the chart.

Example:

```
//+------------------------------------------------------------------+ //|
//|                             Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
input int ticks_to_close=20;// number of ticks before EA unload
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print(TimeCurrent(),": " ,__FUNCTION__," reason code = ",reason);
//--- "clear" comment
   Comment("");
//---
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
   static int tick_counter=0;
//---
   tick_counter++;
   Comment("\nBefore unloading expert advisor ",__FILE__," left",
           (ticks_to_close-tick_counter)," ticks");
//--- before
   if(tick_counter>=ticks_to_close)
     {
      ExpertRemove();
      Print(TimeCurrent(),": ",__FUNCTION__," expert advisor will be unloa
     }
   Print("tick_counter =",tick_counter);
//---
  }
//+------------------------------------------------------------------+
```

See also

Programs running, Client terminal events

GetPointer

The function returns the object pointer.

```
void*  GetPointer(    any_class anyobject      // object of any class
   );
```

Parameters

anyobject

[in] Object of any class.

Return Value

The function returns the object pointer.

Note

Only class objects have pointers. Instances of structures and simple-type variables can't have pointers. The class object not created using the new() operator, but, e.g., automatically created in the array of objects, still has a pointer. But this pointer will be of the automatic type POINTER_AUTOMATIC, therefore the delete() operator can't be applied to it. Aside from that, the type pointer doesn't differ from dynamic pointers of the POINTER_AUTOMATIC type.

Since variables of structure types and simple types do not have pointers, it's prohibited to apply the GetPointer() function to them. It's also prohibited to pass the pointer as a function argument. In all these cases the compiler will notify an error.

An attempt to call an incorrect pointer causes the critical termination of a program. That's why the CheckPointer() function should be called prior to using a pointer. A pointer can be incorrect in the following cases:

· the pointer is equal to NULL;

· the object has been deleted using the delete operator.

This function can be used to check the validity of a pointer. A non-zero value guarantees, that the pointer can be used for accessing.

Example:

```
//+------------------------------------------------------------------+
//|                                           Check_GetPointer.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

//+------------------------------------------------------------------+
//| Class implementing the list element                              |
//+------------------------------------------------------------------+
class CItem
  {
   int               m_id;
   string            m_comment;
   CItem*            m_next;
public:
                     CItem() { m_id=0; m_comment=NULL; m_next=NULL; }
                    ~CItem() { Print("Destructor of ",m_id,
                                     (CheckPointer(GetPointer(this))==POIN
                                     "dynamic":"non-dynamic"); }
   void              Initialize(int id,string comm) { m_id=id; m_comment=c
   void              PrintMe() { Print(__FUNCTION__,":",m_id,m_comment); }
   int               Identifier() { return(m_id); }
   CItem*            Next() {return(m_next); }
   void              Next(CItem *item) { m_next=item; }
  };
//+------------------------------------------------------------------+
//| Simplest class of the list                                       |
//+------------------------------------------------------------------+
class CMyList
  {
   CItem*            m_items;
public:
                     CMyList() { m_items=NULL; }
                    ~CMyList() { Destroy(); }
    bool             InsertToBegin(CItem* item);
    void             Destroy();
  };
//+------------------------------------------------------------------+
//| Inserts list element at the beginning                            |
//+------------------------------------------------------------------+
bool CMyList::InsertToBegin(CItem* item)
  {
   if(CheckPointer(item)==POINTER_INVALID) return(false);
//---
   item.Next(m_items);
   m_items=item;
//---
   return(true);
  }
```

```
//+------------------------------------------------------------------+
//| Deletes the list by deleting elements                            |
//+------------------------------------------------------------------+
void CMyList::Destroy()
  {
//--- service pointer to work in a loop
   CItem* item;
//--- go through the loop and try to delete dynamic pointers
   while(CheckPointer(m_items)!=POINTER_INVALID)
     {
      item=m_items;
      m_items=m_items.Next();
      if(CheckPointer(item)==POINTER_DYNAMIC)
        {
         Print("Dynamyc object ",item.Identifier()," to be deleted");
         delete (item);
        }
      else Print("Non-dynamic object ",item.Identifier()," cannot be delet
     }
//---
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   CMyList list;
   CItem   items[10];
   CItem*  item;
//--- create and add into the list a dynamic object pointer
   item=new CItem;
   if(item!=NULL)
     {
      item.Initialize(100,"dynamic");
      item.PrintMe();
      list.InsertToBegin(item);
     }
//--- add automatic pointers into the list
   for(int i=0; i<10; i++)
     {
      items[i].Initialize(i,"automatic");
      items[i].PrintMe();
      item=GetPointer(items[i]);
      if(CheckPointer(item)!=POINTER_INVALID)
         list.InsertToBegin(item);
     }
//--- add one more dynamic object pointer at the list beginning
```

```
   item=new CItem;
   if(item!=NULL)
     {
      item.Initialize(200,"dynamic");
      item.PrintMe();
      list.InsertToBegin(item);
     }
//--- delete all the list elements
   list.Destroy();
//--- all the list elements will be deleted after the script is over
//--- see the Experts tab in the terminal
  }
```

See also

Object Pointers, Checking the Object Pointer, Object Delete Operator delete

GetTickCount

The GetTickCount() function returns the number of milliseconds that elapsed since the system start.

```
uint  GetTickCount();
```

Return Value

Value of uint type.

Note

Counter is limited by the restrictions of the system timer. Time is stored as an unsigned integer, so it's overfilled every 49.7 days if a computer works uninterruptedly.

Example:

```
#define MAX_SIZE 40 //+---------------------------------------------------
//| Script for measuring computation time of 40 Fibonacci numbers    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Remember the initial value
   uint start=GetTickCount();
//--- A variable for getting the next number in the Fibonacci series
   long fib=0;
//--- In loop calculate the specified amount of numbers from Fibonacci ser
   for(int i=0;i<MAX_SIZE;i++) fib=TestFibo(i);
//--- Get the spent time in milliseconds
   uint time=GetTickCount()-start;
//--- Output a message to the Experts journal
   PrintFormat("Calculating %d first Fibonacci numbers took %d ms",MAX_SIZ
//--- Script completed
   return;
  }
//+------------------------------------------------------------------+
//| Function for getting Fibonacci number by its serial number       |
//+------------------------------------------------------------------+
long TestFibo(long n)
  {
//--- The first member of the Fibonacci series
   if(n<2) return(1);
//--- All other members are calculated by the following formula
   return(TestFibo(n-2)+TestFibo(n-1));
  }
```

See also

Date and Time

GetMicrosecondCount

The GetMicrosecondCount() function returns the number of microseconds that have elapsed since the start of MQL program.

```
ulong  GetMicrosecondCount();
```

Return Value

Value of ulong type.

Example:

```
//+------------------------------------------------------------------+ //|
//+------------------------------------------------------------------+
void Test()
  {
   int    res_int=0;
   double res_double=0;
//---
   for(int i=0;i<10000;i++)
     {
      res_int+=i*i;
      res_int++;
      res_double+=i*i;
      res_double++;
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   uint   ui=0,ui_max=0,ui_min=INT_MAX;
   ulong  ul=0,ul_max=0,ul_min=INT_MAX;
//--- number of measurements
   for(int count=0;count<1000;count++)
     {
      uint  ui_res=0;
      ulong ul_res=0;
      //---
      for(int n=0;n<2;n++)
        {
         //--- select measurement type
         if(n==0) ui=GetTickCount();
         else     ul=GetMicrosecondCount();
         //--- execute code
         Test();
         //--- add measurement result (depending on type)
         if(n==0) ui_res+=GetTickCount()-ui;
         else     ul_res+=GetMicrosecondCount()-ul;
        }
      //--- calculate minimum and maximum time for both measurements
      if(ui_min>ui_res) ui_min=ui_res;
      if(ui_max<ui_res) ui_max=ui_res;
      if(ul_min>ul_res) ul_min=ul_res;
      if(ul_max<ul_res) ul_max=ul_res;
     }
//---
   Print("GetTickCount error(msec): ",ui_max-ui_min);
   Print("GetMicrosecondCount error(msec): ",DoubleToString((ul_max-ul_min
  }
```

See also

[Date and Time](#)

MessageBox

It creates and shows a message box and manages it. A message box contains a message and header, any combination of predefined signs and command buttons.

```
int  MessageBox(    string  text,              // message text
   string  caption=NULL,      // box header
   int     flags=0            // defines set of buttons in the box
   );
```

Parameters

text

[in] Text, containing message to output.

caption=NULL

[in] Optional text to be displayed in the box header. If the parameter is empty, Expert Advisor name is shown in the box header.

flags=0

[in] Optional flags defining appearance and behavior of a message box. Flags can be a combination of a special group of flags.

Return Value

If the function is successfully performed, the returned value is one of values of MessageBox() return codes.

Note

The function can't be called from custom indicators, because indicators are executed in the interface thread and shouldn't slow it down.

MessageBox() function does not work in the Strategy Tester.

PeriodSeconds

This function returns number of seconds in a period.

```
int  PeriodSeconds(    ENUM_TIMEFRAMES  period=PERIOD_CURRENT      // chart
   );
```

Parameters

period=PERIOD_CURRENT

[in] Value of a chart period from the enumeration ENUM_TIMEFRAMES. If the parameter isn't specified, it returns the number of seconds of the current chart period, at which the program runs.

Return Value

Number of seconds in a selected period.

See also

_Period, Chart timeframes, Date and Time, Visibility of objects

PlaySound

It plays a sound file.

```
bool  PlaySound(    string  filename       // file name
   );
```

Parameters

filename

[in] Path to a sound file. If filename=NULL, the playback is stopped.

Return Value

true if the file is found, otherwise - false.

Note

The file must be located in terminal_directory\Sounds or its sub-directory. Only WAV files are played.

Call of PlaySound() with NULL parameter stops playback.

PlaySound() function does not work in the Strategy Tester.

See also

Resources

Print

It enters a message in the Expert Advisor log. Parameters can be of any type.

```
void  Print(    argument,      // first value
   ...            // next values
   );
```

Parameters

...

[in] Any values separated by commas. The number of parameters cannot exceed 64.

Note

Arrays cannot be passed to the Print() function. Arrays must be input element-by-element.

Data of double type are shown with the accuracy of up to 16 digits after a decimal point, and can be output either in traditional or in scientific format, depending on what entry will be more compact. Data of float type are output with 5 digits after a decimal point. To output real numbers with another accuracy or in a predefined format, use the PrintFormat() function.

Data of bool type are output as "true" or "false" lines. Dates are shown as YYYY.MM.DD HH:MI:SS. To show data in another format, use TimeToString(). Data of color type are returned either as R,G,B line or as a color name, if this color is present in the color set.

Print() function does not work during optimization in the Strategy Tester.

Example:

```
void OnStart()
  {
//--- Output DBL_MAX using Print(), this is equivalent to PrintFormat(%%.1
   Print("---- how DBL_MAX looks like -----");
   Print("Print(DBL_MAX)=",DBL_MAX);
//--- Now output a DBL_MAX number using PrintFormat()
   PrintFormat("PrintFormat(%%.16G,DBL_MAX)=%.16G",DBL_MAX);
//--- Output to the Experts journal
// Print(DBL_MAX)=1.797693134862316e+308
// PrintFormat(%.16G,DBL_MAX)=1.797693134862316E+308

//--- See how float is output
   float c=(float)M_PI; // We should explicitly cast to the target type
   Print("c=",c, "    Pi=",M_PI, "    (float)M_PI=",(float)M_PI);
```

```
// c=3.14159    Pi=3.141592653589793    (float)M_PI=3.14159

//--- Show what can happen with arithmetic operations with real types
   double a=7,b=200;
   Print("---- Before arithmetic operations");
   Print("a=",a,"   b=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- Divide a by b (7/200)
   a=a/b;
//--- Now emulate restoring a value in the b variable
   b=7.0/a; // It is expected that b=7.0/(7.0/200.0)=>7.0/7.0*200.0=200 -
//--- Output the newly calculated value of b
   Print("----- After arithmetic operations");
   Print("Print(b)=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- Output to the Experts journal
// Print(b)=200.0
// Print(DoubleToString(b,16))=199.9999999999999716 (see that b is no more

//--- Create a very small value epsilon=1E-013
   double epsilon=1e-13;
   Print("---- Create a very small value");
   Print("epsilon=",epsilon); // Get epsilon=1E-013
//--- Now subtract epsilon from b and again output the value to the Expert
   b=b-epsilon;
//--- Use two ways
   Print("---- After subtracting epsilon from the b variable");
   Print("Print(b)=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- Output to the Experts journal
// Print(b)=199.9999999999999  (now the value of b after subtracting epsil
// Print(DoubleToString(b,16))=199.9999999999998578
//    (now the value of b after subtracting epsilon cannot be rounded to 2
  }
```

See also

Alert(), Comment(), DoubleToString(), StringFormat()

PrintFormat

It formats and enters sets of symbols and values in the Expert Advisor log in accordance with a preset format.

```
void  PrintFormat(    string format_string,    // format string
   ...                          // values of simple types
   );
```

Parameters

format_string

[in] A format string consists of simple symbols, and if the format string is followed by arguments, it also contains format specifications.

...

[in] Any values of simple types separated by commas. Total number of parameters can't exceed 64 including the format string.

Return Value

String.

Note

PrintFormat() function does not work during optimization in the Strategy Tester.

The number, order and type of parameters must exactly match the set of qualifiers, otherwise the print result is undefined. Instead of PrintFormat() you can use printf().

If the format string is followed by parameters, this string must contain format specifications that denote output format of these parameters. Specification of format always starts with the percent sign (%).

A format string is read from left to right. When the first format specification is met (if there is any), the value of the first parameter after the format string is transformed and output according to the preset specification. The second format specification calls transformation and output of the second parameter, and so on till the format string end.

The format specification has the following form:

%[flags][width][.precision][{h | l | ll | I32 | I64}]type

Each field of the format specification is either a simple symbol, or a number denoting a simple format option. The simplest format specification contains only the percent sign (%) and a symbol defining the type of the output parameter (for example, %s). If you need to output the percent sign in the format string, use the format specification %%.

flags

Flag	Description	Default Behavior
(minus)	Left justification within the set width	Right justification
+ (plus)	Output of the + or - sign for values of sign types	The sign is shown only if the value is negative
0 (zero)	Zeroes are added before an output value within the preset width. If **0** flag is specified with an integer format (**i**, **u**, **x**, **X**, **o**, **d**) and accuracy specification is set (for example, %04.d), then 0 is ignored.	Nothing is added
space	A space is shown before an output value, if it is a sign and positive value	Spaces aren't inserted
#	If used together with the format **o**, **x** or **X**, then before the output value 0, 0x or 0X is added respectively.	Nothing is added
	If used together with the format **e**, **E**, **a** or **A**, value is always shown with a decimal point.	Decimal point is shown only if there is a non-zero fractional part.
	If used together with the format **g** or **G**, flag defines presence of a decimal point in the output value and prevents the cutting off of leading zeroes. Flag # is ignored when used together with formats **c**, **d**, **i**, **u**, **s**.	Decimal point is shown only if there is a non-zero fractional part. Leading zeroes are cut off.

width

A non-negative decimal number that sets the minimal number of output symbols of the formatted value. If the number of output symbols is less than the specified width, the corresponding number of spaces is added from the left or right depending on the alignment (flag). If there is flag zero (0), the corresponding number of zeroes is added before the output value. If the number of output symbols is greater than the specified width, the output value is never cut off.

If an asterisk (*) is specified as width, value of int type must be indicated in the corresponding place of the list of passed parameters. It will be used for specifying width of the output value.

precision

A non-negative decimal number that sets the output accuracy - number of digits after a decimal point. As distinct from width specification, accuracy specification can cut off the part of fractional type with or without rounding.

The use of accuracy specification is different for different format types.

Types	Description	Default Behavior
a, A	Accuracy specification sets the number of digits after a decimal point.	Default accuracy 6.
c, C	Not used	
d, i, u, o, x, X	Sets minimal number of output digits. If number of digits in a corresponding parameter is less than this accuracy, zeroes are added to the left of the output value. The output value isn't cut off, if the number of output digits is larger than the specified accuracy.	Default accuracy 1.
e, E, f	Sets number of output digits after a decimal point. The last digit is rounded off.	Default accuracy 6. If set accuracy is 0 or decimal part is absent, the decimal point is not shown.
g, G	Sets maximal number of meaningful numbers.	6 meaningful numbers are output.
s, S	Sets number of output symbols of a string. If the string length exceeds the accuracy, the string is cut off.	The whole string is output.

h | l | ll | I32 | I64

Specification of data sizes, passed as a parameter.

Parameter Type	Used Prefix	Joint Specifier of Type
int	l (lower case	d, i, o, x, or X

	L)	
uint	l (lower case L)	o, u, x, or X
long	ll (two lower case L)	d, i, o, x, or X
short	h	d, i, o, x, or X
ushort	h	o, u, x, or X
int	I32	d, i, o, x, or X
uint	I32	o, u, x, or X
long	I64	d, i, o, x, or X
ulong	I64	o, u, x, or X

type

Type specifier is the only obligatory field for formatted output.

Symbol	Type	Output Format
c	int	Symbol of short type (Unicode)
C	int	Symbol of char type (ANSI)
d	int	Signed decimal integer
i	int	Signed decimal integer
o	int	Unsigned octal integer
u	int	Unsigned decimal integer
x	int	Unsigned hexadecimal integer, using "abcdef"
X	int	Unsigned hexadecimal integer, using "ABCDEF"
e	double	A real value in the format [-] d.dddde[sign] ddd, where d - one decimal digit, dddd - one or more decimal digits, ddd - a three-digit number that determines the size of the exponent, sign - plus or minus
E	double	Similar to the format of e, except that the sign of exponent is output by upper case letter (**E** instead of e)
f	double	A real value in the format [-] dddd.dddd, where dddd - one or more decimal digits. Number of displayed digits before the decimal point depends on the size of number value. Number of digits after the decimal point depends on the required accuracy.

g	double	A real value output in **f** or **e** format depending on what output is more compact.
G	double	A real value output in **F** or **E** format depending on what output is more compact.
a	double	A real number in format [−]0xh.hhhh **p**±dd, where h.hhhh mantissa in the form of hexadecimal digits, using "abcdef", dd - One or more digits of exponent. Number of decimal places is determined by the accuracy specification
A	double	A real number in format [−]0xh.hhhh **P**±dd, where h.hhhh mantissa in the form of hexadecimal digits, using "ABCDEF", dd - One or more digits of exponent. Number of decimal places is determined by the accuracy specification
s	string	String output

Instead of PrintFormat() you can use printf().

Example:

```
void OnStart()
  {
//--- trade server name
   string server=AccountInfoString(ACCOUNT_SERVER);
//--- account number
   int login=(int)AccountInfoInteger(ACCOUNT_LOGIN);
//--- long value output
   long leverage=AccountInfoInteger(ACCOUNT_LEVERAGE);
   PrintFormat("%s %d: leverage = 1:%I64d",
               server,login,leverage);
//--- account currency
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- double value output with 2 digits after the decimal point
   double equity=AccountInfoDouble(ACCOUNT_EQUITY);
   PrintFormat("%s %d: account equity = %.2f %s",
               server,login,equity,currency);
//--- double value output with mandatory output of the +/- sign
   double profit=AccountInfoDouble(ACCOUNT_PROFIT);
   PrintFormat("%s %d: current result for open orders = %+.2f %s",
               server,login,profit,currency);
//--- double value output with variable number of digits after the decimal
   double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
   string format_string=StringFormat("%%s: point value  = %%.%df",_Digits)
   PrintFormat(format_string,_Symbol,point_value);
//--- int value output
   int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
   PrintFormat("%s: current spread in points = %d ",
               _Symbol,spread);
//--- double value output in the scientific (floating point) format with 1
   PrintFormat("DBL_MAX = %.17e",DBL_MAX);
//--- double value output in the scientific (floating point) format with 1
   PrintFormat("EMPTY_VALUE = %.17e",EMPTY_VALUE);
//--- output using PrintFormat() with default accuracy
   PrintFormat("PrintFormat(EMPTY_VALUE) = %e",EMPTY_VALUE);
//--- simple output using Print()
   Print("Print(EMPTY_VALUE) = ",EMPTY_VALUE);
/* execution result
   MetaQuotes-Demo 4236774: leverage = 1:100
   MetaQuotes-Demo 4236774: account equity = 9998.49 USD
   MetaQuotes-Demo 4236774: current result for open orders = -1.51 USD
   EURJPY: point value  = 0.001
   EURJPY: current spread in points = 15
   DBL_MAX = 1.79769313486231570e+308
   EMPTY_VALUE = 2.14748364700000000e+009
   PrintFormat(EMPTY_VALUE) = 2.147484e+009
   Print(EMPTY_VALUE) = 2147483647
*/
  }
```

See also

StringFormat(), DoubleToString(), Real types (double, float)

ResetLastError

Sets the value of the predefined variable _LastError into zero.

```
void  ResetLastError();
```

Return Value

No return value.

Note

The GetLastError() function zero the _LastError variable.

ResourceCreate

Creates an image resource based on a data set. There are two variants of the function: **Creating a resource based on a file**

```
bool  ResourceCreate(
   const string      resource_name,       // Resource name
   const string      path                 // A relative path to the file
   );
```

Creating a resource based on the array of pixels

```
bool  ResourceCreate(
   const string      resource_name,       // Resource name
   const uint&       data[],              // Data set as an array
   uint              img_width,           // The width of the image resour
   uint              img_height,          // The height of the image resou
   uint              data_xoffset,        // The horizontal rightward offs
   uint              data_yoffset,        // The vertical downward offset
   uint              data_width,          // The total width of the image
   ENUM_COLOR_FORMAT color_format        // Color processing method
   );
```

Parameters

resource_name

[in] Resource name.

path

[in] Relative path to the file, containing the resource data. If the path is started from "\" (written as "\\"), it is assumed that file path is relative to **terminal_data_folder**\MQL4\, otherwise it is assumed that file is specified relative to .EX4 program folder.

data[][]

[in] A one-dimensional or two-dimensional array for creating a complete image.

img_width

[in] The width of the rectangular image area in pixels to be placed in the resource in the form of an image. It cannot be greater than the *data_width* value.

img_height

[in] The height of the rectangular image area in pixels to be placed in the resource in the form of an image.

data_xoffset

[in] The horizontal rightward offset of the rectangular area of the image.

data_yoffset

[in] The vertical downward offset of the rectangular area of the image.

data_width

[in] Required only for one-dimensional arrays. It denotes the full width of the image from the data set. If *data_width*=0, it is assumed to be equal to *img_width*. For two-dimensional arrays the parameter is ignored and is assumed to be equal to the second dimension of the *data[]* array.

color_format

[in] Color processing method, from a value from the ENUM_COLOR_FORMAT enumeration.

Return Value

Returns true if successful, otherwise false. To get information about the error call the GetLastError() function. The following errors may occur:

- 4015 ERR_RESOURCE_NAME_DUPLICATED (identical names of the dynamic and the static resource)
- 4016 ERR_RESOURCE_NOT_FOUND (the resource is not found)
- 4017 ERR_RESOURCE_UNSUPPORTED_TYPE (this type of resource is not supported)
- 4018 ERR_RESOURCE_NAME_IS_TOO_LONG (the name of the resource is too long)

Note

If the second version of the function is called for creating the same resource with different width, height and shift parameters, it does not create a new resource, but simply updates the existing one.

The first version of the function is used for uploading images and sounds from files, and the second version is used only for the dynamic creation of images.

Images must be in the BMP format with a color depth of 24 or 32 bits. Sounds can only be in the WAV format. The size of the resource should not exceed 16 Mb.

ENUM_COLOR_FORMAT

Identifier	Description

COLOR_FORMAT_XRGB_NOALPHA	The component of the alpha channel is ignored
COLOR_FORMAT_ARGB_RAW	Color components are not handled by the terminal (must be correctly set by the user)
COLOR_FORMAT_ARGB_NORMALIZE	Color components are handled by the terminal

See also

Resources, ObjectCreate(), ObjectSetString(), OBJPROP_BMPFILE

ResourceFree

The function deletes dynamically created resource (freeing the memory allocated for it).

```
bool  ResourceFree(    const string  resource_name       // resource name
   );
```

Parameters

resource_name

[in] Resource name should start with "::".

Returned value

True if successful, otherwise false. To get information about the error, call the GetLastError() function.

Note

ResourceFree() allows mql4 application developers to manage memory consumption when actively working with resources. Graphical objects bound to the resource being deleted from the memory will be displayed correctly after its deletion. However, newly created graphical objects (OBJ_BITMAP and OBJ_BITMAP_LABEL) will not be able to use the deleted resource.

The function deletes only dynamic resources created by the program.

See also

Resources, ObjectCreate(), PlaySound(), ObjectSetString(), OBJPROP_BMPFILE

ResourceReadImage

The function reads data from the graphical resource created by ResourceCreate() function or saved in EX4 file during compilation.

```
bool  ResourceReadImage(    const string      resource_name,       // grap
   uint&             data[],              // array for receiving data from
   uint&             width,               // for receiving the image width
   uint&             height               // for receiving the image heigh
   );
```

Parameters

resource_name

[in] Name of the graphical resource containing an image. To gain access to its own resources, the name is used in brief form "::resourcename". If we download a resource from a compiled EX4 file, the full name should be used with the path relative to MQL4 directory, file and resource names "path\\filename.ex4::resourcename".

data[][]

[in] One- or two-dimensional array for receiving data from the graphical resource.

img_width

[out] Graphical resource image width in pixels.

img_height

[out] Graphical resource image height in pixels.

Returned value

true if successful, otherwise false. To get information about the error, call the GetLastError() function.

Note

If *data[]* array is then to be used for creating a graphical resource, COLOR_FORMAT_ARGB_NORMALIZE or COLOR_FORMAT_XRGB_NOALPHA color formats should be used.

If *data[]* array is two-dimensional and its second dimension is less than X(width) graphical resource size, ResourceReadImage() function returns false and reading is not performed. But if the resource exists, actual image size is returned to width and height parameters. This will allow making another attempt to receive data from the resource.

See also

Resource, ObjectCreate(), ObjectSetString(), OBJPROP_BMPFILE

ResourceSave

Saves a resource into the specified file.

```
bool  ResourceSave(    const string  resource_name,      // Resource name
   const string  file_name             // File name
   );
```

Parameters

resource_name

[in] The name of the resource, must start with "::".

file_name

[in] The name of the file relative to MQL4\Files.

Return Value

true in case of success, otherwise false. For the error information call GetLastError().

Note

The function always overwrites a file and creates all the required intermediate directories in the file name if necessary.

See also

Resources, ObjectCreate(), PlaySound(), ObjectSetString(), OBJPROP_BMPFILE

SetUserError

Sets the predefined variable _LastError into the value equal to ERR_USER_ERROR_FIRST + user_error.

```
void  SetUserError(    ushort user_error    // error number
   );
```

Parameters

user_error

[in] Error number set by a user.

Return Value

No return value.

Note

After an error has been set using the SetUserError(user_error) function, GetLastError() returns value equal to ERR_USER_ERROR_FIRST + user_error.

Example:

```
void OnStart()
  {
//--- set error number 65537=(ERR_USER_ERROR_FIRST +1)
   SetUserError(1);
//--- get last error code
   Print("GetLastError = ",GetLastError());
//--- Result
//--- GetLastError = 65537
  }
```

SendFTP

Sends a file at the address, specified in the setting window of the "FTP" tab.

```
bool  SendFTP(    string  filename,          // file to be send by ftp
   string  ftp_path=NULL      // ftp catalog
   );
```

Parameters

filename

[in] Name of sent file.

ftp_path=NULL

[in] FTP catalog. If a directory is not specified, directory described in settings is used.

Return Value

In case of failure returns 'false'.

Note

Sent file must be located in the folder *terminal_directory\MQL4\files* or its subfolders. Sending isn't performed if FTP address and/or access password are not specified in settings.

SendFTP() function does not work in the Strategy Tester.

SendNotification

Sends push notifications to the mobile terminals, whose MetaQuotes IDs are specified in the "Notifications" tab.

```
bool  SendNotification(    string  text          // Text of the notificati
   );
```

Parameters

text

[in] The text of the notification. The message length should not exceed 255 characters.

Return Value

true if a notification has been successfully sent from the terminal; in case of failure returns false. When checking after a failed push of notification, GetLastError () may return one of the following errors:

- 4250 ERR_NOTIFICATION_SEND_FAILED,
- 4251 ERR_NOTIFICATION_WRONG_PARAMETER,
- 4252 ERR_NOTIFICATION_WRONG_SETTINGS,
- 4253 ERR_NOTIFICATION_TOO_FREQUENT.

Note

Strict use restrictions are set for the SendNotification() function: no more than 2 calls per second and not more than 10 calls per minute. Monitoring the frequency of use is dynamic. The function can be disabled in case of the restriction violation.

SendNotification() function does not work in the Strategy Tester.

SendMail

Sends an email at the address specified in the settings window of the "Email" tab.

```
bool  SendMail(    string  subject,       // header
   string  some_text      // email text
   );
```

Parameters

subject

[in] Email header.

some_text

[in] Email body.

Return Value

true if an email is put into the send queue, otherwise - false.

Note

Sending can be prohibited in settings, email address can be omitted as well. For the error information call GetLastError().

SendMail() function does not work in the Strategy Tester.

Sleep

The function suspends execution of the current Expert Advisor or script within a specified interval.

```
void  Sleep(    int   milliseconds      // interval
   );
```

Parameters

milliseconds

[in] Delay interval in milliseconds.

Return Value

No return value.

Note

The Sleep() function can't be called for custom indicators, because indicators are executed in the interface thread and must not slow down it. The function has the built-in check of EA halt flag every 0.1 seconds.

Sleep() function does not suspend execution of the Expert Advisor in the Strategy Tester.

TerminalClose

The function commands the terminal to complete operation.

```
bool  TerminalClose(    int ret_code      // closing code of the client te
   );
```

Parameters

ret_code

[in] Return code, returned by the process of the client terminal at the operation completion.

Return Value

The function returns true on success, otherwise - false.

Note

The TerminalClose() function does not stop the terminal immediately, it just commands the terminal to complete its operation.

The code of an Expert Advisor that called TerminalClose() must have all arrangements for the immediate completion (e.g. all previously opened files must be closed in the normal mode). Call of this function must be followed by the return operator.

The *ret_code* parameter allows indicating the necessary return code for analyzing reasons of the program termination of the terminal operation when starting it from the command prompt.

Example:

```
//--- input parameters
input int   tiks_before=500; // number of ticks till termination
input int   pips_to_go=15;   // distance in pips
input int   seconds_st=50;   // number of seconds given to the Expert Advis
//--- globals
datetime    launch_time;
int         tick_counter=0;
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print(__FUNCTION__," reason code = ",reason);
   Comment("");
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
   static double first_bid=0.0;
   MqlTick       tick;
   double        distance;
//---
   SymbolInfoTick(_Symbol,tick);
   tick_counter++;
   if(first_bid==0.0)
     {
      launch_time=tick.time;
      first_bid=tick.bid;
      Print("first_bid =",first_bid);
      return;
     }
//--- price distance in pips
   distance=(tick.bid-first_bid)/_Point;
//--- show a notification to track the EA operation
   string comm="From the moment of start:\r\n\x25CF elapsed seconds: "+
               IntegerToString(tick.time-launch_time)+" ;"+
               "\r\n\x25CF ticks received: "+(string)tick_counter+" ;"+
               "\r\n\x25CF price went in points: "+StringFormat("%G",dista
   Comment(comm);
//--- section for checking condition to close the terminal
   if(tick_counter>=tiks_before)
      TerminalClose(0);    // exit by tick counter
   if(distance>pips_to_go)
      TerminalClose(1);    // go up by the number of pips equal to pips_t
   if(distance<-pips_to_go)
      TerminalClose(-1);   // go down by the number of pips equal to pips_
   if(tick.time-launch_time>seconds_st)
      TerminalClose(100);  // termination by timeout
//---
  }
```

See also

Program running, Execution errors, Reasons for deinitialization

TesterStatistics

The function returns the value of the specified statistical parameter calculated based on testing results.

```
double  TesterStatistics(    ENUM_STATISTICS statistic_id      // ID
   );
```

Parameters

statistic_id

[in] The ID of the statistical parameter from the ENUM_STATISTICS enumeration.

Return Value

The value of the statistical parameter from testing results.

Note

The function can be called inside OnTester() or OnDeinit() in the tester. In other cases the result is undefined.

WebRequest

The function sends an HTTP request to a specified server. The function has two versions:

1. **Sending simple requests** of type "key=value" using the header Content-Type: application/x-www-form-urlencoded.

```
int  WebRequest(    const string      method,          // HTTP method
   const string      url,             // URL
   const string      cookie,          // cookie
   const string      referer,         // referer
   int               timeout,         // timeout
   const char        &data[],         // the array of the HTTP message bc
   int               data_size,       // data[] array size in bytes
   char              &result[],       // an array containing server respc
   string            &result_headers  // headers of server response
   );
```

2. **Sending a request of any type** specifying the custom set of headers for a more flexible interaction with various Web services.

```
int  WebRequest(
   const string      method,          // HTTP method
   const string      url,             // URL
   const string      headers,         // headers
   int               timeout,         // timeout
   const char        &data[],         // the array of the HTTP message bc
   char              &result[],       // an array containing server respc
   string            &result_headers  // headers of server response
   );
```

Parameters

method

[in] HTTP method.

url

[in] URL.

headers

[in] Request headers of type "key: value", separated by a line break "\r\n".

cookie

[in] Cookie value.

referer

[in] Value of the Referer header of the HTTP request.

timeout

[in] Timeout in milliseconds.

data[]

[in] Data array of the HTTP message body.

data_size

[in] Size of the data[] array.

result[]

[out] An array containing server response data.

result_headers

[out] Server response headers.

Returned value

HTTP server response code or -1 for an error.

Note

To use the WebRequest() function, add the addresses of the required servers in the list of allowed URLs in the "Expert Advisors" tab of the "Options" window. Server port is automatically selected on the basis of the specified protocol - 80 for "http://" and 443 for "https://".

The WebRequest() function is synchronous, which means its breaks the program execution and waits for the response from the requested server. Since the delays in receiving a response can be large, the function is not available for calls from the indicators, because indicators run in a common thread shared by all indicators and charts on one symbol. Indicator performance delay on one of the charts of a symbol may stop updating of all charts of the same symbol.

The function can be called only from Expert Advisors and scripts, as they run in their own execution threads. If you try to call the function from an indicator, GetLastError() will return error 4060 "Function is not allowed for call".

WebRequest() cannot be executed in the Strategy Tester.

An example of using the first version of the WebRequest () **function:**

```
void OnStart()
  {
   string cookie=NULL,headers;
   char post[],result[];
   int res;
//--- to enable access to the server, you should add URL "https://www.goog
//--- in the list of allowed URLs (Main Menu->Tools->Options, tab "Expert
   string google_url="https://www.google.com/finance";
//--- Reset the last error code
   ResetLastError();
//--- Loading a html page from Google Finance
   int timeout=5000; //--- Timeout below 1000 (1 sec.) is not enough for s
   res=WebRequest("GET",google_url,cookie,NULL,timeout,post,0,result,heade
//--- Checking errors
   if(res==-1)
     {
      Print("Error in WebRequest. Error code  =",GetLastError());
      //--- Perhaps the URL is not listed, display a message about the nec
      MessageBox("Add the address '"+google_url+"' in the list of allowed
     }
   else
     {
      //--- Load successfully
      PrintFormat("The file has been successfully loaded, File size =%d by
      //--- Save the data to a file
      int filehandle=FileOpen("GoogleFinance.htm",FILE_WRITE|FILE_BIN);
      //--- Checking errors
      if(filehandle!=INVALID_HANDLE)
        {
         //--- Save the contents of the result[] array to a file
         FileWriteArray(filehandle,result,0,ArraySize(result));
         //--- Close the file
         FileClose(filehandle);
        }
      else Print("Error in FileOpen. Error code=",GetLastError());
     }
  }
```

An example of using the second version of the WebRequest() function:

```
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
#property script_show_inputs
#property description "Sample script posting a user message "
#property description "on the wall on mql5.com"
```

```
input string InpLogin   ="";             //Your MQL5.com account
input string InpPassword="";             //Your account password
input string InpFileName="EURUSDM5.png"; //An image in folder MQL5/Files/
input string InpFileType="image/png";    //Correct mime type of the image
//+------------------------------------------------------------------+
//| Posting a message with an image on the wall at mql5.com          |
//+------------------------------------------------------------------+
bool PostToNewsFeed(string login,string password,string text,string filena
  {
   int    res;     // To receive the operation execution result
   char   data[];  // Data array to send POST requests
   char   file[];  // Read the image here
   string str="Login="+login+"&Password="+password;
   string auth,sep="-------Jyecslin9mp8RdKV"; // multipart data separator
//--- A file is available, try to read it
   if(filename!=NULL && filename!="")
     {
      res=FileOpen(filename,FILE_READ|FILE_BIN);
      if(res<0)
        {
         Print("Error opening the file \""+filename+"\"");
         return(false);
        }
      //--- Read file data
      if(FileReadArray(res,file)!=FileSize(res))
        {
         FileClose(res);
         Print("Error reading the file \""+filename+"\"");
         return(false);
        }
      //---
      FileClose(res);
     }
//--- Create the body of the POST request for authorization
   ArrayResize(data,StringToCharArray(str,data,0,WHOLE_ARRAY,CP_UTF8)-1);
//--- Resetting error code
   ResetLastError();
//--- Authorization request
   res=WebRequest("POST","https://www.mql5.com/en/auth_login",NULL,0,data,
//--- If authorization failed
   if(res!=200)
     {
      Print("Authorization error #"+(string)res+", LastError="+(string)Get
      return(false);
     }
//--- Read the authorization cookie from the server response header
```

```
   res=StringFind(str,"Set-Cookie: auth=");
//--- If cookie not found, return an error
   if(res<0)
     {
      Print("Error, authorization data not found in the server response (c
      return(false);
     }
//--- Remember the authorization data and form the header for further requ
   auth=StringSubstr(str,res+12);
   auth="Cookie: "+StringSubstr(auth,0,StringFind(auth,";")+1)+"\r\n";
//--- If there is a data file, send it to the server
   if(ArraySize(file)!=0)
     {
      //--- Form the request body
      str="--"+sep+"\r\n";
      str+="Content-Disposition: form-data; name=\"attachedFile_imagesLoa
      str+="Content-Type: "+filetype+"\r\n\r\n";
      res =StringToCharArray(str,data);
      res+=ArrayCopy(data,file,res-1,0);
      res+=StringToCharArray("\r\n--"+sep+"--\r\n",data,res-1);
      ArrayResize(data,ArraySize(data)-1);
      //--- Form the request header
      str=auth+"Content-Type: multipart/form-data; boundary="+sep+"\r\n";
      //--- Reset error code
      ResetLastError();
      //--- Request to send an image file to the server
      res=WebRequest("POST","https://www.mql5.com/upload_file",str,0,data,
      //--- check the request result
      if(res!=200)
        {
         Print("Error sending a file to the server #"+(string)res+", LastE
         return(false);
        }
      //--- Receive a link to the image uploaded to the server
      str=CharArrayToString(data);
      if(StringFind(str,"{\"Url\":\"")==0)
        {
         res     =StringFind(str,"\"",8);
         filename=StringSubstr(str,8,res-8);
         //--- If file uploading fails, an empty link will be returned
         if(filename=="")
           {
            Print("File sending to server failed");
            return(false);
           }
        }
     }
```

```
//--- Create the body of a request to post an image on the server
   str ="--"+sep+"\r\n";
   str+="Content-Disposition: form-data; name=\"content\"\r\n\r\n";
   str+=text+"\r\n";
//--- The languages in which the post will be available on mql5.com
   str+="--"+sep+"\r\n";
   str+="Content-Disposition: form-data; name=\"AllLanguages\"\r\n\r\n";
   str+="on\r\n";
//--- If the picture has been uploaded on the server, pass its link
   if(ArraySize(file)!=0)
     {
      str+="--"+sep+"\r\n";
      str+="Content-Disposition: form-data; name=\"attachedImage_0\"\r\n\r
      str+=filename+"\r\n";
     }
//--- The final string of the multipart request
   str+="--"+sep+"--\r\n";
//--- Out the body of the POST request together in one string
   StringToCharArray(str,data,0,WHOLE_ARRAY,CP_UTF8);
   ArrayResize(data,ArraySize(data)-1);
//--- Prepare the request header
   str=auth+"Content-Type: multipart/form-data; boundary="+sep+"\r\n";
//--- Request to post a message on the user wall at mql5.com
   res=WebRequest("POST","https://www.mql5.com/ru/users/"+login+"/wall",st
//--- Return true for successful execution
   return(res==200);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Post a message on mql5.com, including an image, the path to which is
   PostToNewsFeed(InpLogin,InpPassword,"Checking the expanded version of W
                  "(This message has been posted by the WebRequest.mq5 scr
  }
//+------------------------------------------------------------------+
```

ZeroMemory

The function resets a variable passed to it by reference.

```
void  ZeroMemory(    void & variable       // reset variable
   );
```

Parameters

variable

[in] [out] Variable passed by reference, you want to reset (initialize by zero values).

Return Value

No return value.

Note

If the function parameter is a string, the call will be equivalent to indicating NULL as its value.
For simple types and their arrays, as well as for structures/classes consisting of such types, this is a simple reset.
For objects containing strings and dynamic arrays, ZeroMemory() is called for each element.
For any arrays not protected by the const modifier, this is the zeroing of all elements.
For arrays of complex objects, ZeroMemory() is called for each element.

ZeroMemory() can't be applied to classes with protected members or inheritance.

Group of Functions for Working with Arrays

Arrays are allowed to be maximum four-dimensional. Each dimension is indexed from 0 to *dimension_size-1*. In a particular case of a one-dimensional array of 50 elements, calling of the first element will appear as array[0], of the last one - as array[49].

Function	Action
ArrayBsearch	Returns index of the first found element in the first array dimension
ArrayCopy	Copies one array into another
ArrayCompare	Returns the result of comparing two arrays of simple types or custom structures without complex objects
ArrayFree	Frees up buffer of any dynamic array and sets the size of the zero dimension in 0.
ArrayGetAsSeries	Checks direction of array indexing
ArrayInitialize	Sets all elements of a numeric array into a single value
ArrayFill	Fills an array with the specified value
ArrayIsSeries	Checks whether an array is a timeseries
ArrayIsDynamic	Checks whether an array is dynamic
ArrayMaximum	Search for an element with the maximal value
ArrayMinimum	Search for an element with the minimal value
ArrayRange	Returns the number of elements in the specified dimension of the array
ArrayResize	Sets the new size in the first dimension of the array
ArraySetAsSeries	Sets the direction of array indexing
ArraySize	Returns the number of elements in the array
ArraySort	Sorting of numeric arrays by the first dimension
ArrayCopyRates	Copies rates to the two-dimensional array from chart RateInfo array returns copied bars amount
ArrayCopySeries	Copies a series array to another one and returns the count of the copied elements
ArrayDimension	Returns the multidimensional array rank

ArrayBsearch

Searches for a specified value in a multidimensional numeric array sorted in the ascending order. The search is performed in the first dimension taking into account the AS_SERIES flag.

For searching in an array of double type

```
int  ArrayBsearch(    const double&   array[],              // array for
   double             value,                 // what is searched for
   int                count=WHOLE_ARRAY,     // count of elements to search f
   int                start=0,               // starting position
   int                direction=MODE_ASCEND  // search direction
   );
```

For searching in an array of float type

```
int  ArrayBsearch(
   const float&       array[],               // array for search
   float              value,                 // what is searched for
   int                count=WHOLE_ARRAY,     // count of elements to search f
   int                start=0,               // starting position
   int                direction=MODE_ASCEND  // search direction
   );
```

For searching in an array of long type

```
int  ArrayBsearch(
   const long&       array[],               // array for search
   long              value,                 // what is searched for
   int               count=WHOLE_ARRAY,     // count of elements to search fo
   int               start=0,               // starting position
   int               direction=MODE_ASCEND  // search direction
   );
```

For searching in an array of int type

```
int  ArrayBsearch(
   const int&       array[],               // array for search
   int              value,                 // what is searched for
   int              count=WHOLE_ARRAY,     // count of elements to search for
   int              start=0,               // starting position
   int              direction=MODE_ASCEND  // search direction
   );
```

For searching in an array of short type

```
int  ArrayBsearch(
   const short&    array[],                // array for search
   short           value,                  // what is searched for
   int             count=WHOLE_ARRAY,      // count of elements to search fo
   int             start=0,                // starting position
   int             direction=MODE_ASCEND   // search direction
   );
```

For searching in an array of char type

```
int  ArrayBsearch(
   const char&     array[],                // array for search
   char            value,                  // what is searched for
   int             count=WHOLE_ARRAY,      // count of elements to search fo
   int             start=0,                // starting position
   int             direction=MODE_ASCEND   // search direction
   );
```

Parameters

array[]

[in] Numeric array for search.

value

[in] Value for search.

count=WHOLE_ARRAY

[in] Count of elements to search for. By default, it searches in the whole array.

start=0

[in] Starting index to search for. By default, the search starts at the first element.

direction=MODE_ASCEND

[in] Search direction. It can be any of the following values:

MODE_ASCEND searching in forward direction,
MODE_DESCEND searching in backward direction.

Return Value

The function returns index of a found element. If the wanted value isn't found, the function returns the index of an element nearest in value.

Note

Binary search processes only sorted arrays. To sort numeric arrays use the ArraySort() function.

Example:

```
   datetime daytimes[];
   int      shift=10,dayshift;
   // All the Time[] series 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_DESCEN
      if(Period()<PERIOD_D1) dayshift++;
     }
   Print(TimeToStr(Time[shift])," corresponds to ",dayshift," day bar oper
         TimeToStr(daytimes[dayshift]));
```

ArrayCopy

It copies an array into another one.

```
int  ArrayCopy(    void&         dst_array[],         // destination array
   const void&   src_array[],         // source array
   int           dst_start=0,         // index starting from which write in
   int           src_start=0,         // first index of a source array
   int           count=WHOLE_ARRAY    // number of elements
   );
```

Parameters

dst_array[]

[out] Destination array

src_array[]

[in] Source array

dst_start=0

[in] Starting index from the destination array. By default, start index is 0.

src_start=0

[in] Starting index for the source array. By default, start index is 0.

count=WHOLE_ARRAY

[in] Number of elements that should be copied. By default, the whole array is copied (count=WHOLE_ARRAY).

Return Value

It returns the number of copied elements.

Note

If count<=0 or count>src_size-src_start, all the remaining array part is copied. Arrays are copied from left to right. For series arrays, the starting position is correctly defined adjusted for copying from left to right. If an array is copied to itself, the result is undefined.

If arrays are of different types, during copying it tries to transform each element of a source array into the type of the destination array. A string array can be copied into a string array only. Array of classes and structures containing objects that require initialization aren't copied. An array of structures can be copied into an array of the same type only.

For static and dynamic arrays (except for class and structure members), the size of a destination array is automatically increased to the amount of copied data (if the latter exceeds the array size).

Example:

```
void OnStart()
  {
//---
   int src_data[10];
   for (int i=0; i<ArraySize(src_data); i++) src_data[i]=i;
   int dst_data[];
   //--- copy data to dst_data[]
   ArrayCopy(dst_data,src_data,0,0,WHOLE_ARRAY);
   //--- print copied data[]
   PrintFormat("Copied array size=%d",ArraySize(dst_data));
   for (int i=0; i<ArraySize(dst_data); i++) PrintFormat("index=%d, value=
  }
```

ArrayCompare

The function returns the result of comparing two arrays of the same type. It can be used to compare arrays of simple types or custom structures without complex objects, that is the custom structures that do not contain strings, dynamic arrays, classes and other structures with complex objects.

```
int  ArrayCompare(   const void&   array1[],            // first array
   const void&  array2[],            // second array
   int          start1=0,            // initial offset in the first array
   int          start2=0,            // initial offset in the second array
   int          count=WHOLE_ARRAY    // number of elements for comparison
   );
```

Parameters

array1[]

[in] First array.

array2[]

[in] Second array.

start1=0

[in] The element's initial index in the first array, from which comparison starts. The default start index - 0.

start2=0

[in] The element's initial index in the second array, from which comparison starts. The default start index - 0.

count=WHOLE_ARRAY

[in] Number of elements to be compared. All elements of both arrays participate in comparison by default (count=WHOLE_ARRAY).

Returned value

- -1, if array1[] less than array2[]
- 0, if array1[] equal to array2[]
- 1, if array1[] more than array2[]
- -2, if an error occurs due to incompatibility of the types of compared arrays or if start1, start2 or count values lead to falling outside the array.

Note

The function will not return 0 (the arrays will not be considered equal) if the arrays differ in size and count=WHOLE_ARRAY for the case when one array is a faithful subset of another one. In this case, the result of comparing the sizes of that arrays will be returned: -1, if the size of array1[] is less than the size of array2[] , otherwise 1.

ArrayFree

It frees up a buffer of any dynamic array and sets the size of the zero dimension to 0.

```
void  ArrayFree(    void&  array[]      // array
   );
```

Parameters

array[]

[in] Dynamic array.

Return Value

No return value.

Note

The need for using ArrayFree() function may not appear too often considering that all used memory is freed at once and main work with the arrays comprises the access to the indicator buffers. The sizes of the buffers are automatically managed by the terminal's executive subsystem.

In case it is necessary to manually manage the memory in complex dynamic environment of the application, ArrayFree() function allows users to free the memory occupied by the already unnecessary dynamic array explicitly and immediately.

Example:

```
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
#include <Controls\Label.mqh>
#include <Controls\ComboBox.mqh>
//--- predefined constants
#define X_START 0
#define Y_START 0
#define X_SIZE 280
#define Y_SIZE 300
//+------------------------------------------------------------------+
//| Dialog class for working with memory                             |
//+------------------------------------------------------------------+
class CMemoryControl : public CAppDialog
  {
private:
   //--- array size
   int               m_arr_size;
   //--- arrays
   char              m_arr_char[];
   int               m_arr_int[];
   float             m_arr_float[];
   double            m_arr_double[];
   long              m_arr_long[];
   //--- labels
   CLabel            m_lbl_memory_physical;
   CLabel            m_lbl_memory_total;
   CLabel            m_lbl_memory_available;
   CLabel            m_lbl_memory_used;
   CLabel            m_lbl_array_size;
   CLabel            m_lbl_array_type;
   CLabel            m_lbl_error;
   CLabel            m_lbl_change_type;
   CLabel            m_lbl_add_size;
   //--- buttons
   CButton           m_button_add;
   CButton           m_button_free;
   //--- combo boxes
   CComboBox         m_combo_box_step;
   CComboBox         m_combo_box_type;
   //--- current value of the array type from the combo box
   int               m_combo_box_type_value;

public:
                     CMemoryControl(void);
                    ~CMemoryControl(void);
   //--- class object creation method
   virtual bool      Create(const long chart,const string name,const int s
   //--- handler of chart events
   virtual bool      OnEvent(const int id,const long &lparam,const double

protected:
   //--- create labels
   bool              CreateLabel(CLabel &lbl,const string name,const int x
   //--- create control elements
   bool              CreateButton(CButton &button,const string name,const
   bool              CreateComboBoxStep(void);
   bool              CreateComboBoxType(void);
   //--- event handlers
   void              OnClickButtonAdd(void);
   void              OnClickButtonFree(void);
   void              OnChangeComboBoxType(void);
   //--- methods for working with the current array
   void              CurrentArrayFree(void);
   bool              CurrentArrayAdd(void);
```

```
  };
//+------------------------------------------------------------------+
//| Free memory of the current array                                 |
//+------------------------------------------------------------------+
void CMemoryControl::CurrentArrayFree(void)
  {
//--- reset array size
   m_arr_size=0;
//--- free the array
   if(m_combo_box_type_value==0)
      ArrayFree(m_arr_char);
   if(m_combo_box_type_value==1)
      ArrayFree(m_arr_int);
   if(m_combo_box_type_value==2)
      ArrayFree(m_arr_float);
   if(m_combo_box_type_value==3)
      ArrayFree(m_arr_double);
   if(m_combo_box_type_value==4)
      ArrayFree(m_arr_long);
  }
//+------------------------------------------------------------------+
//| Attempt to add memory for the current array                      |
//+------------------------------------------------------------------+
bool CMemoryControl::CurrentArrayAdd(void)
  {
//--- exit if the size of the used memory exceeds the size of the physical
   if(TerminalInfoInteger(TERMINAL_MEMORY_PHYSICAL)/TerminalInfoInteger(TE
      return(false);
//--- attempt to allocate memory according to the current type
   if(m_combo_box_type_value==0 && ArrayResize(m_arr_char,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==1 && ArrayResize(m_arr_int,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==2 && ArrayResize(m_arr_float,m_arr_size)==-1)
      return(false);
   if(m_combo_box_type_value==3 && ArrayResize(m_arr_double,m_arr_size)==-1
      return(false);
   if(m_combo_box_type_value==4 && ArrayResize(m_arr_long,m_arr_size)==-1)
      return(false);
//--- memory allocated
   return(true);
  }
//+------------------------------------------------------------------+
//| Handling events                                                  |
//+------------------------------------------------------------------+
EVENT_MAP_BEGIN(CMemoryControl)
ON_EVENT(ON_CLICK,m_button_add,OnClickButtonAdd)
ON_EVENT(ON_CLICK,m_button_free,OnClickButtonFree)
ON_EVENT(ON_CHANGE,m_combo_box_type,OnChangeComboBoxType)
EVENT_MAP_END(CAppDialog)
//+------------------------------------------------------------------+
//| Constructor                                                      |
//+------------------------------------------------------------------+
CMemoryControl::CMemoryControl(void)
  {
  }
//+------------------------------------------------------------------+
//| Destructor                                                       |
//+------------------------------------------------------------------+
CMemoryControl::~CMemoryControl(void)
  {
  }
//+------------------------------------------------------------------+
//| Class object creation method                                     |
//+------------------------------------------------------------------+
bool CMemoryControl::Create(const long chart,const string name,const int s
                            const int x1,const int y1,const int x2,const i
  {
//--- create base class object
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- prepare strings for labels
   string str_physical="Memory physical = "+(string)TerminalInfoInteger(TE
   string str_total="Memory total = "+(string)TerminalInfoInteger(TERMINAL
   string str_available="Memory available = "+(string)TerminalInfoInteger(
   string str_used="Memory used = "+(string)TerminalInfoInteger(TERMINAL_M
//--- create labels
   if(!CreateLabel(m_lbl_memory_physical,"physical_label",X_START+10,Y_STA
      return(false);
   if(!CreateLabel(m_lbl_memory_total,"total_label",X_START+10,Y_START+30,
      return(false);
   if(!CreateLabel(m_lbl_memory_available,"available_label",X_START+10,Y_S
      return(false);
   if(!CreateLabel(m_lbl_memory_used,"used_label",X_START+10,Y_START+80,st
      return(false);
   if(!CreateLabel(m_lbl_array_type,"type_label",X_START+10,Y_START+105,"A
      return(false);
   if(!CreateLabel(m_lbl_array_size,"size_label",X_START+10,Y_START+130,"A
      return(false);
   if(!CreateLabel(m_lbl_error,"error_label",X_START+10,Y_START+155,"",12,
      return(false);
   if(!CreateLabel(m_lbl_change_type,"change_type_label",X_START+10,Y_STAR
      return(false);
   if(!CreateLabel(m_lbl_add_size,"add_size_label",X_START+10,Y_START+210,
      return(false);
//--- create control elements
   if(!CreateButton(m_button_add,"add_button",X_START+15,Y_START+245,"Add"
      return(false);
   if(!CreateButton(m_button_free,"free_button",X_START+75,Y_START+245,"Fr
      return(false);
   if(!CreateComboBoxType())
      return(false);
   if(!CreateComboBoxStep())
      return(false);
//--- initialize the variable
   m_arr_size=0;
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Create the button                                                |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateButton(CButton &button,const string name,const
                                  const int y,const string str,const int f
                                  const int clr)
  {
//--- create the button
   if(!button.Create(m_chart_id,name,m_subwin,x,y,x+50,y+20))
      return(false);
//--- text
   if(!button.Text(str))
      return(false);
//--- font size
   if(!button.FontSize(font_size))
      return(false);
//--- label color
   if(!button.Color(clr))
      return(false);
//--- add the button to the control elements
   if(!Add(button))
      return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Create a combo box for the array size                            |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateComboBoxStep(void)
  {
//--- create the combo box
   if(!m_combo_box_step.Create(m_chart_id,"step_combobox",m_subwin,X_START-
      return(false);
//--- add elements to the combo box
   if(!m_combo_box_step.ItemAdd("100 000",100000))
      return(false);
   if(!m_combo_box_step.ItemAdd("1 000 000",1000000))
      return(false);
   if(!m_combo_box_step.ItemAdd("10 000 000",10000000))
      return(false);
   if(!m_combo_box_step.ItemAdd("100 000 000",100000000))
      return(false);
//--- set the current combo box element
   if(!m_combo_box_step.SelectByValue(1000000))
      return(false);
//--- add the combo box to control elements
   if(!Add(m_combo_box_step))
      return(false);
//--- successful execution
   return(true);
  }
//+------------------------------------------------------------------+
//| Create a combo box for the array type                            |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateComboBoxType(void)
  {
//--- create the combo box
   if(!m_combo_box_type.Create(m_chart_id,"type_combobox",m_subwin,X_START-
      return(false);
//--- add elements to the combo box
   if(!m_combo_box_type.ItemAdd("char",0))
      return(false);
   if(!m_combo_box_type.ItemAdd("int",1))
      return(false);
   if(!m_combo_box_type.ItemAdd("float",2))
      return(false);
   if(!m_combo_box_type.ItemAdd("double",3))
      return(false);
   if(!m_combo_box_type.ItemAdd("long",4))
      return(false);
//--- set the current combo box element
   if(!m_combo_box_type.SelectByValue(3))
      return(false);
//--- store the current combo box element
   m_combo_box_type_value=3;
//--- add the combo box to control elements
   if(!Add(m_combo_box_type))
      return(false);
//--- successful execution
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Create a label                                                   |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateLabel(CLabel &lbl,const string name,const int x
                                 const int y,const string str,const int fo
                                 const int clr)
  {
//--- create a label
   if(!lbl.Create(m_chart_id,name,m_subwin,x,y,0,0))
      return(false);
//--- text
   if(!lbl.Text(str))
      return(false);
//--- font size
   if(!lbl.FontSize(font_size))
      return(false);
//--- color
   if(!lbl.Color(clr))
      return(false);
//--- add the label to control elements
   if(!Add(lbl))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Handler of clicking "Add" button event                           |
//+------------------------------------------------------------------+
void CMemoryControl::OnClickButtonAdd(void)
  {
//--- increase the array size
   m_arr_size+=(int)m_combo_box_step.Value();
//--- attempt to allocate memory for the current array
   if(CurrentArrayAdd())
     {
      //--- memory allocated, display the current status on the screen
      m_lbl_memory_available.Text("Memory available = "+(string)TerminalIn
      m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(
      m_lbl_array_size.Text("Array size = "+IntegerToString(m_arr_size));
      m_lbl_error.Text("");
     }
   else
     {
      //--- failed to allocate memory, display the error message
      m_lbl_error.Text("Array is too large, error!");
      //--- return the previous array size
      m_arr_size-=(int)m_combo_box_step.Value();
     }
  }
//+------------------------------------------------------------------+
//| Handler of clicking "Free" button event                          |
//+------------------------------------------------------------------+
void CMemoryControl::OnClickButtonFree(void)
  {
//--- free the memory of the current array
   CurrentArrayFree();
//--- display the current status on the screen
   m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoI
   m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TER
   m_lbl_array_size.Text("Array size = 0");
   m_lbl_error.Text("");
  }
//+------------------------------------------------------------------+
//| Handler of the combo box change event                            |
//+------------------------------------------------------------------+
void CMemoryControl::OnChangeComboBoxType(void)
  {
//--- check if the array's type has changed
   if(m_combo_box_type.Value()!=m_combo_box_type_value)
     {
      //--- free the memory of the current array
      OnClickButtonFree();
      //--- work with another array type
      m_combo_box_type_value=(int)m_combo_box_type.Value();
      //--- display the new array type on the screen
      if(m_combo_box_type_value==0)
         m_lbl_array_type.Text("Array type = char");
      if(m_combo_box_type_value==1)
         m_lbl_array_type.Text("Array type = int");
      if(m_combo_box_type_value==2)
         m_lbl_array_type.Text("Array type = float");
      if(m_combo_box_type_value==3)
         m_lbl_array_type.Text("Array type = double");
      if(m_combo_box_type_value==4)
         m_lbl_array_type.Text("Array type = long");
     }
  }
//--- CMemoryControl class object
CMemoryControl ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- create the dialog
   if(!ExtDialog.Create(0,"MemoryControl",0,X_START,Y_START,X_SIZE,Y_SIZE))
      return(INIT_FAILED);
//--- launch
   ExtDialog.Run();
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   ExtDialog.Destroy(reason);
  }
//+------------------------------------------------------------------+
//| Expert chart event function                                      |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
   ExtDialog.ChartEvent(id,lparam,dparam,sparam);
  }
```

ArrayGetAsSeries

It checks direction of an array index.

```
bool  ArrayGetAsSeries(    const void&  array[]     // array for checking
   );
```

Parameters

array

[in] Checked array.

Return Value

Returns true, if the specified array has the AS_SERIES flag set, i.e. access to the array is performed back to front as in timeseries. A timeseries differs from a usual array in that the indexing of timeseries elements is performed from its end to beginning (from the newest data to old).

Note

To check whether an array belongs to timeseries, use the ArrayIsSeries() function. Arrays of price data passed as input parameters into the OnCalculate() function do not obligatorily have the indexing direction the same as in timeseries. The necessary indexing direction can be set using the ArraySetAsSeries() function.

Example:

```
#property description "Indicator calculates absolute values of the differe
#property description "Open and Close or High and Low prices displaying th
#property description "as a histrogram."
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 1
//--- input parameters
input bool InpAsSeries=true; // Indexing direction in the indicator buffer
input bool InpPrices=true;   // Calculation prices (true - Open,Close; fal
//--- indicator buffer
double ExtBuffer[];
//+------------------------------------------------------------------+
//| Calculating indicator values                                     |
//+------------------------------------------------------------------+
void CandleSizeOnBuffer(const int rates_total,const int prev_calculated,
                        const double &first[],const double &second[],doubl
  {
//--- start variable for calculation of bars
```

```
   int start_index=prev_calculated;
//--- work at the last bar if the indicator values have already been calcu
   if(prev_calculated>0)
      start_index--;
//--- define indexing direction in arrays
   bool as_series_first=ArrayGetAsSeries(first);
   bool as_series_second=ArrayGetAsSeries(second);
   bool as_series_buffer=ArrayGetAsSeries(buffer);
//--- replace indexing direction with direct one if necessary
   if(as_series_first)
      ArraySetAsSeries(first,false);
   if(as_series_second)
      ArraySetAsSeries(second,false);
   if(as_series_buffer)
      ArraySetAsSeries(buffer,false);
//--- calculate indicator values
   for(int i=start_index;i<rates_total;i++)
      buffer[i]=MathAbs(first[i]-second[i]);
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- bind indicator buffers
   SetIndexBuffer(0,ExtBuffer);
//--- set indexing element in the indicator buffer
   ArraySetAsSeries(ExtBuffer,InpAsSeries);
//--- check for what prices the indicator is calculated
   if(InpPrices)
     {
      //--- Open and Close prices
      IndicatorShortName("BodySize");
      //--- set the indicator color
      SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3,clrOrange);
     }
   else
     {
      //--- High and Low prices
      IndicatorShortName("ShadowSize");
      //--- set the indicator color
      SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3,clrDodgerBlue);
     }
//---
//---
   return(INIT_SUCCEEDED);
  }
```

```
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- calculate the indicator according to the flag value
   if(InpPrices)
      CandleSizeOnBuffer(rates_total,prev_calculated,open,close,ExtBuffer)
   else
      CandleSizeOnBuffer(rates_total,prev_calculated,high,low,ExtBuffer);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Access to timeseries, ArraySetAsSeries

ArrayInitialize

The function initializes a numeric array by a preset value.

For initialization of an array of char type

```
int  ArrayInitialize(    char     array[],     // initialized array
   char     value        // value that will be set
   );
```

For initialization of an array of short type

```
int  ArrayInitialize(
   short   array[],     // initialized array
   short   value        // value that will be set
   );
```

For initialization of an array of int type

```
int  ArrayInitialize(
   int     array[],     // initialized array
   int     value        // value that will be set
   );
```

For initialization of an array of long type

```
int  ArrayInitialize(
   long    array[],     // initialized array
   long    value        // value that will be set
   );
```

For initialization of an array of float type

```
int  ArrayInitialize(
   float   array[],     // initialized array
   float   value        // value that will be set
   );
```

For initialization of an array of double type

```
int  ArrayInitialize(
   double  array[],     // initialized array
   double  value        // value that will be set
   );
```

For initialization of an array of bool type

```
int  ArrayInitialize(
   bool     array[],      // initialized array
   bool     value         // value that will be set
   );
```

For initialization of an array of uint type

```
int  ArrayInitialize(
   uint     array[],      // initialized array
   uint     value         // value that will be set
   );
```

Parameters

array[]

[out] Numeric array that should be initialized.

value

[in] New value that should be set to all array elements.

Return Value

No return value.

Note

The ArrayResize() function allows to set size of an array with a reserve for further expansion without the physical relocation of memory. It is implemented for the better performance, because the operations of memory relocation are reasonably slow.

Initialization of the array using ArrayInitialize(array, init_val) doesn't mean the initialization with the same value of reserve elements allocated for this array. At further expanding of the *array* using the ArrayResize() function, the elements will be added at the end of the array, their values will be undefined and in most cases will not be equal to *init_value*.

Example:

```
void OnStart()
  {
//--- dynamic array
   double array[];
//--- let's set the array size for 100 elements and reserve a buffer for a
   ArrayResize(array,100,10);
//--- initialize the array elements with EMPTY_VALUE=DBL_MAX value
   ArrayInitialize(array,EMPTY_VALUE);
   Print("Values of 10 last elements after initialization");
   for(int i=90;i<100;i++) printf("array[%d] = %G",i,array[i]);
//--- expand the array by 5 elements
   ArrayResize(array,105);
   Print("Values of 10 last elements after ArrayResize(array,105)");
//--- values of 5 last elements are obtained from reserve buffer
   for(int i=95;i<105;i++) printf("array[%d] = %G",i,array[i]);
  }
```

ArrayFill

The function fills an array with the specified value.

```
void  ArrayFill(    void&  array[],       // array
   int     start,         // starting index
   int     count,         // number of elements to fill
   void    value          // value
   );
```

Parameters

array[]

[out] Array of simple type (char, uchar, short, ushort, int, uint, long, ulong, bool, color, datetime, float, double).

start

[in] Starting index. In such a case, specified AS_SERIES flag is ignored.

count

[in] Number of elements to fill.

value

[in] Value to fill the array with.

Returned value

No return value.

Note

When ArrayFill() function is called, normal indexation direction (from left to right) is always implied. It means that the change of the order of access to the array elements using ArraySetAsSeries() function is ignored.

A multidimensional array is shown as one-dimensional when processed by ArrayFill() function. For example, array[2][4] is processed as array[8]. Therefore, you may specify the initial element's index to be equal to 5 when working with this array. Thus, the call of ArrayFill(array, 5, 2, 3.14) for array[2][4] fills array[1][1] and array[1][2] elements with 3.14.

Example:

```
void OnStart()
  {
//--- declare dynamic array
   int a[];
//--- set size
   ArrayResize(a,10);
//--- fill first 5 elements with 123
   ArrayFill(a,0,5,123);
//--- fill next 5 elements with 456
   ArrayFill(a,5,5,456);
//--- show values
   for(int i=0;i<ArraySize(a);i++) printf("a[%d] = %d",i,a[i]);
  }
```

ArrayIsDynamic

The function checks whether an array is dynamic.

```
bool  ArrayIsDynamic(    const void&  array[]    // checked array
   );
```

Parameters

array[]

[in] Checked array.

Return Value

It returns true if the selected array is dynamic, otherwise it returns false.

Example:

```
#property description "This indicator does not calculate values. It makes
#property description "apply the call of ArrayFree() function to three arr
#property description "an indicator buffer. Results are shown in Experts j
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 1
//--- global variables
double ExtDynamic[];   // dynamic array
double ExtStatic[100]; // static array
bool   ExtFlag=true;   // flag
double ExtBuff[];      // indicator buffer
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- allocate memory for the array
   ArrayResize(ExtDynamic,100);
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtBuff);
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- perform a single analysis
   if(ExtFlag)
     {
      //--- attempt to free memory for arrays
      //--- 1. Dynamic array
      Print("+============================+");
      Print("1. Check dynamic array:");
      Print("Size before memory is freed = ",ArraySize(ExtDynamic));
      Print("Is this a dynamic array = ",ArrayIsDynamic(ExtDynamic) ? "Yes
      //--- attempt to free array memory
      ArrayFree(ExtDynamic);
      Print("Size after memory is freed = ",ArraySize(ExtDynamic));
      //--- 2. Static array
      Print("2. Check static array:");
      Print("Size before memory is freed = ",ArraySize(ExtStatic));
      Print("Is this a dynamic array = ",ArrayIsDynamic(ExtStatic) ? "Yes"
      //--- attempt to free array memory
      ArrayFree(ExtStatic);
      Print("Size after memory is freed = ",ArraySize(ExtStatic));
      //--- 3. Indicator buffer
      Print("3. Check indicator buffer:");
      Print("Size before memory is freed = ",ArraySize(ExtBuff));
      Print("Is this a dynamic array = ",ArrayIsDynamic(ExtBuff) ? "Yes" :
      //--- attempt to free array memory
      ArrayFree(ExtBuff);
      Print("Size after memory is freed = ",ArraySize(ExtBuff));
      //--- change the flag value
      ExtFlag=false;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Access to timeseries and indicators

ArrayIsSeries

The function checks whether an array is a timeseries.

```
bool  ArrayIsSeries(    const void&  array[]    // checked array
   );
```

Parameters

array[]

[in] Checked array.

Return Value

It returns true, if a checked array is an array timeseries, otherwise it returns false. Arrays passed as a parameter to the OnCalculate() function must be checked for the order of accessing the array elements by ArrayGetAsSeries().

Example:

```
#property indicator_chart_window
#property indicator_buffers 1
//---- plot Label1
#property indicator_label1  "Label1"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- indicator buffers
double         Label1Buffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   if(ArrayIsSeries(open))
      Print("open[] is timeseries");
   else
      Print("open[] is not timeseries!!!");
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Access to timeseries and indicators

ArrayMaximum

The function searches a maximal element in a one-dimension numeric array.

```
int  ArrayMaximum(    const void&    array[],              // array for searc
   int              count=WHOLE_ARRAY,    // number of checked elements
   int              start=0               // index to start checking with
   );
```

Parameters

array[]

[in] A numeric array, in which search is made.

count=WHOLE_ARRAY

[in] Number of elements for search. By default, searches in the entire array (count=WHOLE_ARRAY).

start=0

[in] Index to start checking with.

Return Value

The function returns an index of a found element taking into account the array serial. In case of failure it returns -1.

Example:

```
void OnStart()
  {
//---
   double num_array[15]={4,1,6,3,19,4,2,6,3,9,4,5,6,3,9};
   int    maxValueIdx=ArrayMaximum(num_array,WHOLE_ARRAY,0);
   Print("Max value = ",num_array[maxValueIdx]," at index=",maxValueIdx);
  }
```

See also

ArrayMinimum()

ArrayMinimum

The function searches a minimal element in a one-dimension numeric array.

```
int  ArrayMinimum(   const void&    array[],             // array for sear
   int           count=WHOLE_ARRAY,   // number of checked elements
   int           start=0              // index to start checking with
   );
```

Parameters

array[]

[in] A numeric array, in which search is made.

count=WHOLE_ARRAY

[in] Number of elements for search. By default, searches in the entire array (count=WHOLE_ARRAY).

start=0

[in] Index to start checking with.

Return Value

The function returns an index of a found element taking into account the array serial. In case of failure it returns -1.

Example:

```
void OnStart()
  {
//---
   double num_array[15]={4,1,6,3,19,4,2,6,3,9,4,5,6,3,9};
   int    minValueIdx=ArrayMinimum(num_array,WHOLE_ARRAY,0);
   Print("Min value = ",num_array[minValueIdx]," at index=",minValueIdx);
  }
```

See also

ArrayMaximum()

ArrayRange

The function returns the number of elements in a selected array dimension.

```
int  ArrayRange(    const void&   array[],      // array for check
   int             rank_index    // index of dimension
   );
```

Parameters

array[]

[in] Checked array.

rank_index

[in] Index of dimension.

Return Value

Number of elements in a selected array dimension.

Note

Since indexes start at zero, the number of the array dimensions is one greater than the index of the last dimension.

Example:

```
void OnStart()
  {
//--- create four-dimensional array
   double array[][5][2][4];
//--- set the size of the zero dimension
   ArrayResize(array,10,10);
//--- print dimensions
   int temp;
   for(int i=0;i<4;i++)
     {
      //--- receive the size of i dimension
      temp=ArrayRange(array,i);
      //--- print
      PrintFormat("dim = %d, range = %d",i,temp);
     }
//--- Result
// dim = 0, range = 10
// dim = 1, range = 5
// dim = 2, range = 2
// dim = 3, range = 4
  }
```

ArrayResize

The function sets a new size for the first dimension

```
int  ArrayResize(    void&  array[],              // array passed by refer
   int    new_size,             // new array size
   int    reserve_size=0        // reserve size value (excess)
   );
```

Parameters

array[]

[out] Array for changing sizes.

new_size

[in] New size for the first dimension.

reserve_size=0

[in] Optional parameter. Distributed size to get reserve.

Return Value

If executed successfully, it returns count of all elements contained in the array after resizing, otherwise, returns -1, and array is not resized.

Note

The function can be applied only to dynamic arrays. It should be noted that you cannot change the size of dynamic arrays assigned as indicator buffers by the SetIndexBuffer() function. For indicator buffers, all operations of resizing are performed by the runtime subsystem of the terminal.

Total amount of elements in the array cannot exceed 2147483647.

With the frequent memory allocation, it is recommended to use a third parameter that sets a reserve to reduce the number of physical memory allocations. All the subsequent calls of ArrayResize do not lead to physical reallocation of memory, but only change the size of the first array dimension within the reserved memory. It should be remembered that the third parameter will be used only during physical memory allocation. For example:

```
ArrayResize(arr,1000,1000);
for(int i=1;i<3000;i++)
   ArrayResize(arr,i,1000);
```

In this case the memory will be reallocated twice, first before entering the 2000-element loop (the array size will be set to 1000), and the second time with i equal to 2000. If we skip the third parameter, there will be 2000 physical reallocations of memory, which will slow down the program.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Counters
   ulong start=GetTickCount();
   ulong now;
   int   count=0;
//--- An array for demonstration of a quick version
   double arr[];
   ArrayResize(arr,100000,100000);
//--- Check how fast the variant with memory reservation works
   Print("--- Test Fast: ArrayResize(arr,100000,100000)");
   for(int i=1;i<=300000;i++)
     {
      //--- Set a new array size specifying the reserve of 100,000 element
      ArrayResize(arr,i,100000);
      //--- When reaching a round number, show the array size and the time
      if(ArraySize(arr)%100000==0)
        {
         now=GetTickCount();
         count++;
         PrintFormat("%d. ArraySize(arr)=%d Time=%d ms",count,ArraySize(ar
         start=now;
        }
     }
//--- Now show, how slow the version without memory reservation is
   double slow[];
   ArrayResize(slow,100000,100000);
//---
   count=0;
   start=GetTickCount();
   Print("---- Test Slow: ArrayResize(slow,100000)");
//---
   for(int i=1;i<=300000;i++)
     {
      //--- Set a new array size, but without the additional reserve
      ArrayResize(slow,i);
      //--- When reaching a round number, show the array size and the time
      if(ArraySize(slow)%100000==0)
        {
         now=GetTickCount();
         count++;
         PrintFormat("%d. ArraySize(slow)=%d Time=%d ms",count,ArraySize(s
         start=now;
        }
     }
  }
//--- A sample result of the script
/*
   Test_ArrayResize (EURUSD,H1)   --- Test Fast: ArrayResize(arr,100000,10
   Test_ArrayResize (EURUSD,H1)   1. ArraySize(arr)=100000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   2. ArraySize(arr)=200000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   3. ArraySize(arr)=300000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   ---- Test Slow: ArrayResize(slow,100000)
   Test_ArrayResize (EURUSD,H1)   1. ArraySize(slow)=100000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   2. ArraySize(slow)=200000 Time=0 ms
   Test_ArrayResize (EURUSD,H1)   3. ArraySize(slow)=300000 Time=228511 ms
*/
```

See also

ArrayInitialize

ArraySetAsSeries

The function sets the AS_SERIES flag to a selected object of a dynamic array, and elements will be indexed like in timeseries.

```
bool  ArraySetAsSeries(    const void&  array[],     // array by reference
   bool           flag          // true denotes reverse order of indexing
   );
```

Parameters

array[]

[in][out] Numeric array to set.

flag

[in] Array indexing direction.

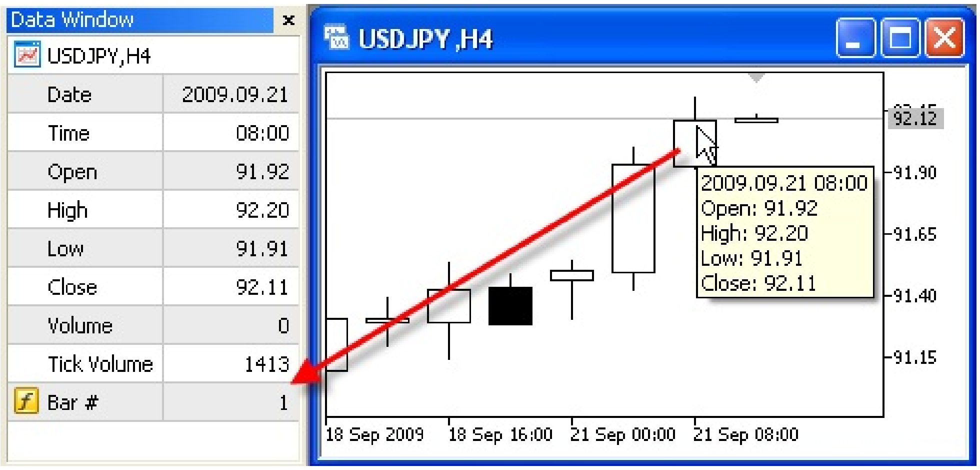
Return Value

The function returns true on success, otherwise - false.

Note

The AS_SERIES flag can't be set for multi-dimensional arrays or static arrays (arrays, whose size in square brackets is preset already on the compilation stage). Indexing in timeseries differs from a common array in that the elements of timeseries are indexed from the end towards the beginning (from the newest to oldest data).

Example: Indicator that shows bar number



Data Window

USDJPY,H4

Date	2009.09.21
Time	08:00
Open	91.92
High	92.20
Low	91.91
Close	92.11
Volume	0
Tick Volume	1413
Bar #	1

```
#property indicator_chart_window
#property indicator_buffers 1
//--- indicator buffers
double          NumerationBuffer[];
```

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,NumerationBuffer,INDICATOR_DATA);
//--- set buffer style
   SetIndexStyle(0,DRAW_LINE,STYLE_SOLID,1,CLR_NONE);
//--- set indexing for the buffer like in timeseries
   ArraySetAsSeries(NumerationBuffer,true);
//--- set accuracy of showing in DataWindow
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//--- how the name of the indicator arry is displayed in DataWindow
   IndicatorShortName("Bar #");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---  we'll store the time of the current zero bar opening
   static datetime currentBarTimeOpen=0;
//--- revert access to array time[] - do it like in timeseries
   ArraySetAsSeries(time,true);
//--- If time of zero bar differs from the stored one
   if(currentBarTimeOpen!=time[0])
     {
     //--- enumerate all bars from the current to the chart depth
      for(int i=rates_total-1;i>=0;i--) NumerationBuffer[i]=i;
      currentBarTimeOpen=time[0];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Access to timeseries, ArrayGetAsSeries

ArraySize

The function returns the number of elements of a selected array.

```
int  ArraySize(    const void&   array[]     // checked array
   );
```

Parameters

array[]

[in] Array of any type.

Return Value

Value of int type.

Note

For a one-dimensional array, the value to be returned by the ArraySize is equal to that of ArrayRange(array,0).

Example:

```
void OnStart()
  {
//--- create arrays
   double one_dim[];
   double four_dim[][10][5][2];
//--- sizes
   int one_dim_size=25;
   int reserve=20;
   int four_dim_size=5;
//--- auxiliary variable
   int size;
//--- allocate memory without backup
   ArrayResize(one_dim,one_dim_size);
   ArrayResize(four_dim,four_dim_size);
//--- 1. one-dimensional array
   Print("+==========================================================+");
   Print("Array sizes:");
   Print("1. One-dimensional array");
   size=ArraySize(one_dim);
   PrintFormat("Zero dimension size = %d, Array size = %d",one_dim_size,si
//--- 2. multidimensional array
   Print("2. Multidimensional array");
   size=ArraySize(four_dim);
   PrintFormat("Zero dimension size = %d, Array size = %d",four_dim_size,s
//--- dimension sizes
   int d_1=ArrayRange(four_dim,1);
   int d_2=ArrayRange(four_dim,2);
   int d_3=ArrayRange(four_dim,3);
   Print("Check:");
   Print("Zero dimension = Array size / (First dimension * Second dimensio
   PrintFormat("%d = %d / (%d * %d * %d)",size/(d_1*d_2*d_3),size,d_1,d_2,
//--- 3. one-dimensional array with memory backup
   Print("3. One-dimensional array with memory backup");
//--- double the value
   one_dim_size*=2;
//--- allocate memory with backup
   ArrayResize(one_dim,one_dim_size,reserve);
//--- print out the size
   size=ArraySize(one_dim);
   PrintFormat("Size with backup = %d, Actual array size = %d",one_dim_siz
  }
```

ArraySort

Sorts numeric arrays by first dimension. The AS_SERIES flag is taken into account in sorting.

```
bool  ArraySort(    void&        array[],                // array for sorti
   int          count=WHOLE_ARRAY,      // count
   int          start=0,                // starting index
   int          direction=MODE_ASCEND   // sort direction
   );
```

Parameters

array[]

[in][out] Numeric array for sorting.

count=WHOLE_ARRAY

[in] Count of elements to sort. By default, it sorts the whole array.

start=0

[in] Starting index to sort. By default, the sort starts at the first element.

direction=MODE_ASCEND

[in] Sort direction. It can be any of the following values:

MODE_ASCEND sort in ascend direction,
MODE_DESCEND sort in descend direction.

Return Value

The function returns true on success, otherwise - false.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- example of sorting of one dimensional array
   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,WHOLE_ARRAY,0,MODE_DESCEND);
//--- now array is sorted 9,6,4,3,1

//--- example of sorting of two dimensional array
   int  DataArray[5][2]={{7,3},{3,1},{57,14},{12,4},{11,1}};
//--- sorting of DataArray[][] by first dimension  (ascending)
   ArraySort(DataArray,WHOLE_ARRAY,0,MODE_ASCEND);
//--- print sorted array
   for(int i=0; i<5; i++)
     {
      string str="index "+IntegerToString(i)+": ";
      for(int j=0; j<2; j++)
        {
         if(j==0) str+="{"; else str+=",";
         str+=IntegerToString(DataArray[i,j]);
         if(j==1) str+="}";
        }
      Print(str);
     }
//--- output
//index 0: {3,1}
//index 1: {7,3}
//index 2: {11,1}
//index 0: {12,4}
//index 4: {57,14}
  }
```

ArrayCopyRates

Copies rates data to the array and returns the amount of bars copied. There are 2 variants of the function:

```
int  ArrayCopyRates(    MqlRates&  rates_array[],    // MqlRates array, pas
   string      symbol=NULL,      // symbol
   int         timeframe=0       // timeframe
   );
```

Copies rates data to the RateInfo[][6] two-dimensional array of double type and returns the amount of bars copied.

```
int  ArrayCopyRates(
   void&      dest_array[][],     // destination array, passed by reference
   string     symbol=NULL,        // symbol
   int        timeframe=0         // timeframe
   );
```

Parameters

rates_array[]

[out] Destination array of MqlRates type.

dest_array[]

[out] Two-dimensional destination array of double type.

symbol=NULL

[in] Symbol name.

timeframe=0

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

Returned value

The function returns copied bars amount, or -1 if failed.

If data (symbol name and/or timeframe differ from the current ones) are requested from another chart, the situation is possible that the corresponding chart was not opened in the client terminal and the necessary data must be requested from the server. In this case, error ERR_HISTORY_WILL_UPDATED (4066 - the requested history data are under updating) will be placed in the last_error variable, and one will has to re-request (see example of ArrayCopySeries()).

Note

This rates array is normally used to pass data to a DLL function.

In the first variant of the function it performs the virtual data copying to the array of MqlRates type. It means that if timeseries data has been updated, rates_array[] array always will refer to the actual data.

In the second variant it performs the real data copying to dest_array[][] array. The destination array will be resized to the size of the timeseries (even if the destination array has been declared as static).

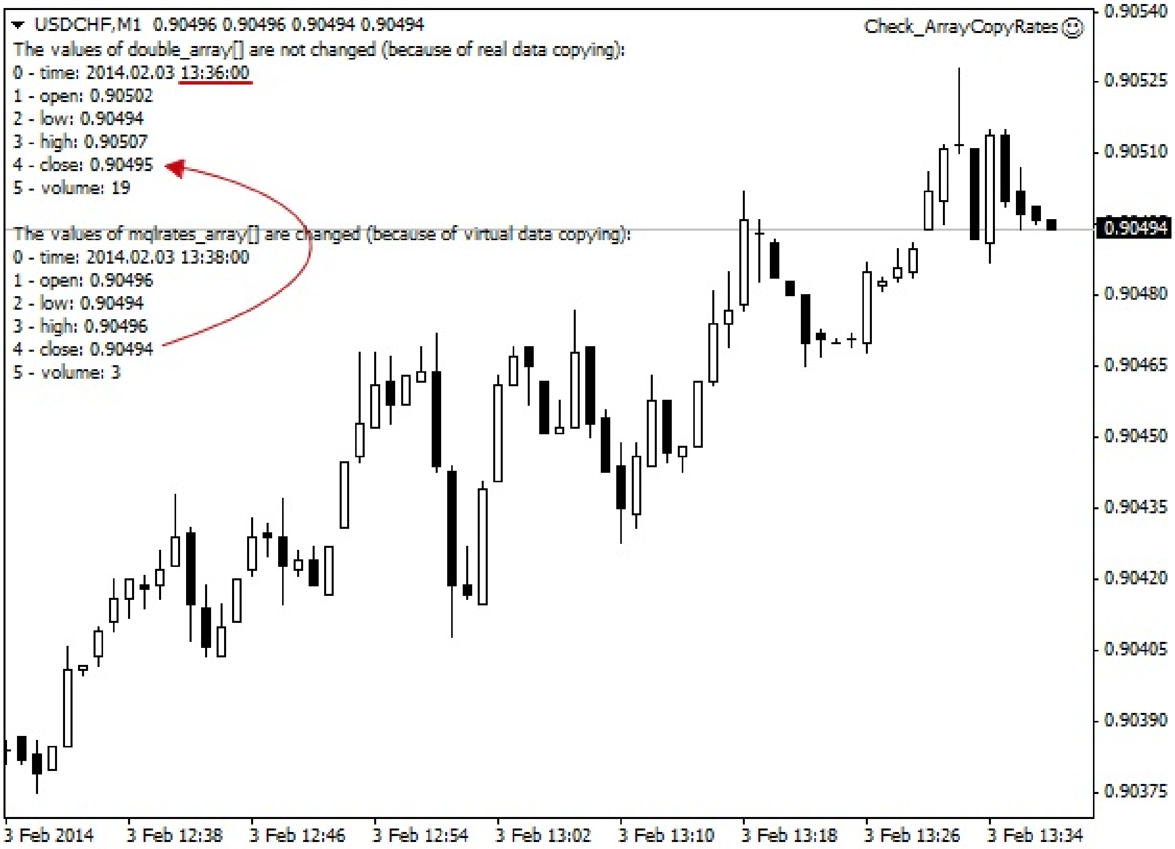
First dimension of RateInfo array contains bars amount, second dimension has 6 elements:

0 - time,
1 - open,
2 - low,
3 - high,
4 - close,
5 - volume.

Example:

```
#property copyright "Copyright © 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
#property description "Expert Advisor displaying two cases of"
#property description "ArrayCopyRates() function call"
//--- destination array for physical copying of historical data
double   double_array[][6];
//--- destination array for logical copying of historical data
MqlRates mqlrates_array[];
//--- first call flag
bool first_call;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- run this EA at chart with M1 timeframe
   if(Period()!=PeriodSeconds(PERIOD_M1)/60)
     {
     Alert("The Expert Advisor must be attached to M1 chart!");
     return(INIT_FAILED);
     }
//--- first call
   first_call=true;
//--- all ok
```

```
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- delete all comments
   Comment("");
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---
   if(first_call)
     {
      //--- copying physically  double_array
      ArrayCopyRates(double_array,NULL,0);
      //--- virtual copying -> mqlrates_array will contain the reference c
      ArrayCopyRates(mqlrates_array,NULL,0);
      //--- cancel first call flag
      first_call=false;
     }
   //--- at each tick print the values of the 0-th array element to see th
   Comment("The values of double_array[] are not changed (because of real
           "0 - time: ",(datetime)double_array[0][0],"\n",
           "1 - open: ",double_array[0][1],"\n"
           "2 - low: ",double_array[0][2],"\n"
           "3 - high: ",double_array[0][3],"\n"
           "4 - close: ",double_array[0][4],"\n"
           "5 - volume: ",DoubleToString(double_array[0][5],0),"\n\n",
           "The values of mqlrates_array[] are changed (because of virtual
           "0 - time: ",mqlrates_array[0].time,"\n",
           "1 - open: ",mqlrates_array[0].open,"\n"
           "2 - low: ",mqlrates_array[0].low,"\n"
           "3 - high: ",mqlrates_array[0].high,"\n"
           "4 - close: ",mqlrates_array[0].close,"\n"
           "5 - volume: ",mqlrates_array[0].tick_volume);
  }
```



```
The values of double_array[] are not changed (because of real data copying):
0 - time: 2014.02.03 13:36:00
1 - open: 0.90502
2 - low: 0.90494
3 - high: 0.90507
4 - close: 0.90495
5 - volume: 19

The values of mqlrates_array[] are changed (because of virtual data copying):
0 - time: 2014.02.03 13:38:00
1 - open: 0.90496
2 - low: 0.90494
3 - high: 0.90496
4 - close: 0.90494
5 - volume: 3
```

ArrayCopySeries

Copies a series array to another one and returns the count of the copied elements.

```
int  ArrayCopySeries(    void&   array[],           // destination array
   int    series_index,       // series array identifier
   string symbol=NULL,        // symbol
   int    timeframe=0         // timeframe
   );
```

Parameters

array[]

[out] Destination array of double type.

series_index

[in] Series array identifier. It can be any of the Series array identifier enumeration values.

symbol

[in] Symbol name.

timeframe

[in] Timeframe. It can be any of Timeframe enumeration values. 0 means the current chart timeframe.

Returned value

The function returns copied elements amount, or -1 if failed.

If data are copied from another chart with different symbol and/or timeframe, it is possible that the necessary data will lack. In this case, error ERR_HISTORY_WILL_UPDATED (4066 - requested history data under updating) will be placed into the last_error variable, and there will be necessary to retry copying after a certain period of time.

Note

There is no real memory allocation for data array and nothing is copied. When such an array is accessed, the access is redirected. Excluded are arrays that are assigned as indexed ones in custom indicators. In this case, data are really copied.

If series_index is MODE_TIME, the array to be passed to the function must be of the datetime type.

Example:

```
datetime daytimes[];
int      shift=10,dayshift,error;
//---- the Time[] array was sroted in the descending order
ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
error=GetLastError();
if(error==4066)
  {
   //---- make two more attempts to read
   for(int i=0;i<2; i++)
     {
      Sleep(5000);
      ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
      //---- check the current daily bar time
      datetime last_day=daytimes[0];
      if(Year()==TimeYear(last_day) && Month()==TimeMonth(last_day) && Day
     }
  }
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
```

ArrayDimension

Returns the multidimensional array rank.

```
int  ArrayDimension(    void&   array[]           // array
   );
```

Parameters

array[]

[in] Array for which the rank will be returned.

Returned value

Rank (dimension) of multidimensional array or -1 if error.

Example:

```
void OnStart()
  {
   int num_array[10][5];
   int dim_size=ArrayDimension(num_array);// dim_size=2
   Print("Dimension of num_array=",dim_size);
  }
```

Conversion Functions

This is a group of functions that provide conversion of data from one format into another.

The NormalizeDouble() function must be specially noted as it provides the necessary accuracy of the price presentation. In trading operations, no unnormalized prices may be used if their accuracy even a digit exceeds that required by the trade server.

Function	Action
CharToString	Converting a symbol code into a one-character string
DoubleToString	Converting a numeric value to a text line with a specified accuracy
EnumToString	Converting an enumeration value of any type to string
NormalizeDouble	Rounding of a floating point number to a specified accuracy
StringToDouble	Converting a string containing a symbol representation of number into number of double type
StringToInteger	Converting a string containing a symbol representation of number into number of int type
StringToTime	Converting a string containing time or date in "yyyy.mm.dd [hh:mi]" format into datetime type
TimeToString	Converting a value containing time in seconds elapsed since 01.01.1970 into a string of "yyyy.mm.dd hh:mi" format
IntegerToString	Converting int into a string of preset length
ShortToString	Converting symbol code (unicode) into one-symbol string
ShortArrayToString	Copying array part into a string
StringToShortArray	Symbol-wise copying a string to a selected part of array of ushort type
CharArrayToString	Converting symbol code (ansi) into one-symbol array
StringToCharArray	Symbol-wise copying a string converted from Unicode to ANSI, to a selected place of array of uchar type
ColorToARGB	Converting color type to uint type to receive ARGB representation of the color.
ColorToString	Converting color value into string as "R,G,B"
StringToColor	Converting "R,G,B" string or string with color name into color type

	value
StringFormat	Converting number into string according to preset format
CharToStr	Conversion of the symbol code into a one-character string
DoubleToStr	Returns text string with the specified numerical value converted into a specified precision format
StrToDouble	Converts string representation of number to double type
StrToInteger	Converts string containing the value character representation into a value of the integer type
StrToTime	Converts string in the format "yyyy.mm.dd hh:mi" to datetime type
TimeToStr	Converts value of datetime type into a string of "yyyy.mm.dd hh:mi" format

See also

Use of a Codepage

CharToString

Converting a symbol code into a one-character string.

```
string  CharToString(    uchar   char_code       // numeric code of symbol
   );
```

Parameters

char_code

[in] Code of ANSI symbol.

Return Value

String with a ANSI symbol.

CharArrayToString

It copies and converts part of array of uchar type into a returned string.

```
string  CharArrayToString(    uchar  array[],              // array
   int    start=0,              // starting position in the array
   int    count=-1,             // number of symbols
   uint   codepage=CP_ACP       // code page
   );
```

Parameters

array[]

[in] Array of uchar type.

start=0

[in] Position from which copying starts. by default 0 is used.

count=-1

[in] Number of array elements for copying. Defines the length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0.

codepage=CP_ACP

[in] The value of the code page. There is a number of built-in constants for the most used code pages.

Return Value

String.

See also

Use of a Codepage

ColorToARGB

The function converts color type into uint type to get ARGB representation of the color. ARGB color format is used to generate a graphical resource, text display, as well as for CCanvas standard library class.

```
uint  ColorToARGB(    color  clr,          // converted color in color for
   uchar  alpha=255     // alpha channel managing color transparency
   );
```

Parameters

clr

[in] Color value in color type variable.

alpha

[in] The value of the alpha channel used to receive the color in ARGB format. The value may be set from 0 (a color of a foreground pixel does not change the display of an underlying one) up to 255 (a color of an underlying pixel is completely replaced by the foreground pixel's one). Color transparency in percentage terms is calculated as (1-alpha/255)*100%. In other words, the lesser value of the alpha channel leads to more transparent color.

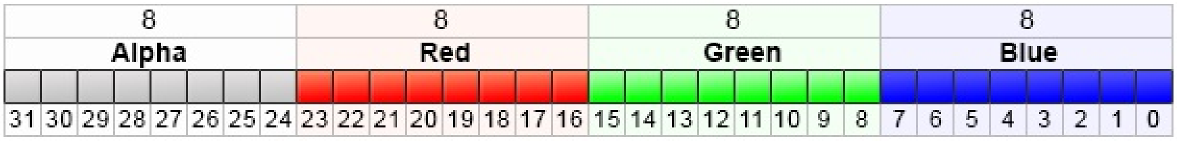
Returned value

Presenting the color in ARGB format where Alfa, Red, Green, Blue (alpha channel, red, green, blue) values are set in series in four uint type bytes.

Note

RGB is a basic and commonly used format for pixel color description on a screen in computer graphics. Names of basic colors are used to set red, green and blue color components. Each component is described by one byte specifying the color saturation in the range of 0 to 255 (0x00 to 0XFF in hexadecimal format). Since the white color contains all colors, it is described as 0xFFFFFF, that is, each one of three components is presented by the maximum value of 0xFF.

However, some tasks require to specify the color transparency to describe the look of an image in case it is covered by the color with some degree of transparency. The concept of alpha channel is introduced for such cases. It is implemented as an additional component of RGB format. ARGB format structure is shown below.



8	8	8	8
Alpha	Red	Green	Blue
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0

ARGB values are typically expressed using hexadecimal format with each pair of digits representing the values of Alpha, Red, Green and Blue channels, respectively. For example, 80FFFF00 color represents 50.2% opaque yellow. Initially, 0x80 sets 50.2% alpha value, as it is 50.2% of 0xFF value. Then, the first FF pair defines the highest value of the red component; the next FF pair is like the previous but for the green component; the final 00 pair represents the lowest value the blue component can have (absence of blue). Combination of green and red colors yields yellow one. If the alpha channel is not used, the entry can be reduced down to 6 RRGGBB digits, this is why the alpha channel values are stored in the top bits of uint integer type.

Depending on the context, hexadecimal digits can be written with '0x' or '#' prefix, for example, 80FFFF00, 0x80FFFF00 or #80FFFF00.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set transparency
   uchar alfa=0x55;  // 0x55 means 55/255=21.6 % of transparency
   //--- derive conversion to ARGB for clrBlue color
   PrintFormat("0x%.8X - clrBlue",clrBlue);
   PrintFormat("0x%.8X - clrBlue ARGB with alfa=0x55 (transparency 21.6%%)
   //--- derive conversion to ARGB for clrGreen color
   PrintFormat("0x%.8X - clrGreen",clrGreen);
   PrintFormat("0x%.8X - clrGreen ARGB with alfa=0x55 (transparency 21.6%%
   //--- derive conversion to ARGB for clrRed color
   PrintFormat("0x%.8X - clrRed",clrRed);
   PrintFormat("0x%.8X - clrRed ARGB with alfa=0x55 (transparency 21.6%%)"
  }
```

See also

Resources, ResourceCreate(), TextOut(), color type, char, short, int and long types

ColorToString

It converts color value into string of "R,G,B" form.

```
string  ColorToString(    color   color_value,     // color value
   bool    color_name        // show color name or not
   );
```

Parameters

color_value

[in] Color value in color type variable.

color_name

[in] Return color name if it is identical to one of predefined color constants.

Return Value

String presentation of color as "R,G,B", where R, G and B are decimal constants from 0 to 255 converted into a string. If the color_name=true parameter is set, it will try to convert color value into color name.

Example:

```
   string clr=ColorToString(C'0,255,0'); // green color
   Print(clr);

   clr=ColorToString(C'0,255,0',true);   // get color constant
   Print(clr);
```

DoubleToString

Converting numeric value into text string.

```
string  DoubleToString(    double  value,       // number
   int      digits=8     // number of digits after decimal point
   );
```

Parameters

value

[in] Value with a floating point.

digits

[in] Accuracy format. If the *digits* value is in the range between 0 and 16, a string presentation of a number with the specified number of digits after the point will be obtained. If the *digits* value is in the range between -1 and -16, a string representation of a number in the scientific format with the specified number of digits after the decimal point will be obtained. In all other cases the string value will contain 8 digits after the decimal point.

Return Value

String containing a symbol representation of a number with the specified accuracy.

Example:

```
   Print("DoubleToString(120.0 + M_PI) : ",DoubleToString(120.0+M_PI));
   Print("DoubleToString(120.0 + M_PI,16) : ",DoubleToString(120.0+M_PI,16
   Print("DoubleToString(120.0 + M_PI,-16) : ",DoubleToString(120.0+M_PI,-
   Print("DoubleToString(120.0 + M_PI,-1) : ",DoubleToString(120.0+M_PI,-1
   Print("DoubleToString(120.0 + M_PI,-20) : ",DoubleToString(120.0+M_PI,-
```

See also

NormalizeDouble, StringToDouble

EnumToString

Converting an enumeration value of any type to a text form.

```
string  EnumToString(    any_enum  value      // any type enumeration value
   );
```

Parameters

value

[in] Any type enumeration value.

Return Value

A string with a text representation of the enumeration. To get the error message call the GetLastError() function.

Note

The function can set the following error values in the _LastError variable:

- ERR_INTERNAL_ERROR error of the execution environment
- ERR_NOT_ENOUGH_MEMORY not enough memory to complete the operation
- ERR_INVALID_PARAMETER can't allow the name of the enumeration value

Example:

```
enum interval  // enumeration of named constants
  {
   month=1,     // one-month interval
   two_months,  // two months
   quarter,     // three months - a quarter
   halfyear=6,  // half a year
   year=12,     // a year - 12 months
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- set the time interval equal to one month
   interval period=month;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- set the time interval equal to a quarter (three months)
   period=quarter;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- set the time interval equal to one year (12 months)
   period=year;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- check how the order type is shown
   ENUM_ORDER_TYPE type=ORDER_TYPE_BUY;
   Print(EnumToString(type)+"="+IntegerToString(type));

//--- check how incorrect values are shown
   type=WRONG_VALUE;
   Print(EnumToString(type)+"="+IntegerToString(type));

// Result:
// month=1
// quarter=3
// year=12
// ORDER_TYPE_BUY=0
// ENUM_ORDER_TYPE::-1=-1
  }
```

See also

Enumerations, Input variables

IntegerToString

This function converts value of integer type into a string of a specified length and returns the obtained string.

```
string  IntegerToString(    long    number,              // number
   int      str_len=0,           // length of result string
   ushort  fill_symbol=' '      // filler
   );
```

Parameters

number

[in] Number for conversion.

str_len=0

[in] String length. If the resulting string length is larger than the specified one, the string is not cut off. If it is smaller, filler symbols will be added to the left.

fill_symbol=' '

[in] Filler symbol. By default it is a space.

Return Value

String.

ShortToString

It converts the symbol code (unicode) into one-symbol string and returns resulting string.

```
string  ShortToString(    ushort  symbol_code        // symbol
   );
```

Parameters

symbol_code

[in] Symbol code. Instead of a symbol code you can use literal string containing a symbol or a literal string with 2-byte hexadecimal code corresponding to the symbol from the Unicode table.

Return Value

String.

ShortArrayToString

It copies part of array into a returned string.

```
string  ShortArrayToString(    ushort   array[],       // array
   int       start=0,       // starting position in the array
   int       count=-1       // number of symbols
   );
```

Parameters

array[]

[in] Array of ushort type (analog of wchar_t type).

start=0

[in] Position, from which copying starts, Default - 0.

count=-1

[in] Number of array elements to copy. Defines the length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0.

Return Value

String.

TimeToString

Converting a value containing time in seconds elapsed since 01.01.1970 into a string of "yyyy.mm.dd hh:mi" format.

```
string  TimeToString(    datetime  value,                          // numb
   int        mode=TIME_DATE|TIME_MINUTES      // output format
   );
```

Parameters

value

[in] Time in seconds from 00:00 1970/01/01.

mode=TIME_DATE|TIME_MINUTES

[in] Additional data input mode. Can be one or combined flag:
TIME_DATE gets result as "yyyy.mm.dd",
TIME_MINUTES gets result as "hh:mi",
TIME_SECONDS gets results as "hh:mi:ss".

Return Value

String.

NormalizeDouble

Rounding floating point number to a specified accuracy.

```
double  NormalizeDouble(    double  value,       // normalized number
   int      digits       // number of digits after decimal point
   );
```

Parameters

value

[in] Value with a floating point.

digits

[in] Accuracy format, number of digits after point (0-8).

Return Value

Value of double type with preset accuracy.

Note

Calculated values of StopLoss, TakeProfit, and values of open prices for pending orders must be normalized with the accuracy, the value of which can be obtained by Digits().

Please note that when output to Journal using the Print() function, a normalized number may contain a greater number of decimal places than you expect. For example, for:

```
   double a=76.671;                 // A normalized number with three decimal
   Print("Print(76.671)=",a);     // Output as is
   Print("DoubleToString(a,8)=",DoubleToString(a,8)); // Output with a pre
```

you will have the following in the terminal:

```
DoubleToString(a,8)=76.67100000
Print(76.671)=76.67100000000001
```

Example:

```
   double pi=M_PI;
   Print("pi = ",DoubleToString(pi,16));

   double pi_3=NormalizeDouble(M_PI,3);
   Print("NormalizeDouble(pi,3) = ",DoubleToString(pi_3,16))
   ;
   double pi_8=NormalizeDouble(M_PI,8);
   Print("NormalizeDouble(pi,8) = ",DoubleToString(pi_8,16));

   double pi_0=NormalizeDouble(M_PI,0);
   Print("NormalizeDouble(pi,0) = ",DoubleToString(pi_0,16));
/*
   Result:
   pi= 3.1415926535897931
   NormalizeDouble(pi,3)= 3.1419999999999999
   NormalizeDouble(pi,8)= 3.1415926499999998
   NormalizeDouble(pi,0)= 3.0000000000000000
*/
```

See also

DoubleToString, Real types (double, float), Reduction of types,

StringToCharArray

Symbol-wise copies a string converted from Unicode to ANSI, to a selected place of array of uchar type. It returns the number of copied elements.

```
int  StringToCharArray(    string  text_string,          // source string
   uchar&   array[],             // array
   int      start=0,             // starting position in the array
   int      count=-1,            // number of symbols
   uint     codepage=CP_ACP      // code page
   );
```

Parameters

text_string

[in] String to copy.

array[]

[out] Array of uchar type.

start=0

[in] Position from which copying starts. Default - 0.

count=-1

[in] Number of array elements to copy. Defines length of a resulting string. Default value is -1, which means copying up to the array end, or till terminating '\0'. Terminating zero will also be copied to the recipient array, in this case the size of a dynamic array can be increased if necessary to the size of the string. If the size of the dynamic array exceeds the length of the string, the size of the array will not be reduced.

codepage=CP_ACP

[in] The value of the code page. For the most-used code pages provide appropriate constants.

Return Value

Number of copied elements.

Note

StringToCharArray() function includes terminating zero. To exclude it specify the string length explicitely:

```
//--- example of copying of string str to array[]
StringToCharArray(str,array,0,StringLen(str));
```

See also

Use of a Codepage

StringToColor

Converting "R,G,B" string or string with color name into color type value.

```
color  StringToColor(    string  color_string      // string representatio
   );
```

Parameters

color_string

[in] String representation of a color of "R,G,B" type or name of one of predefined Web-colors.

Return Value

Color value.

Example:

```
   color str_color=StringToColor("0,127,0");
   Print(str_color);
   Print((string)str_color);
//--- change color a little
   str_color=StringToColor("0,128,0");
   Print(str_color);
   Print((string)str_color);
```

StringToDouble

The function converts string containing a symbol representation of number into number of double type.

```
double  StringToDouble(    string   value       // string
   );
```

Parameters

value

[in] String containing a symbol representation of a number.

Return Value

Value of double type.

StringToInteger

The function converts string containing a symbol representation of number into number of int (integer) type.

```
long  StringToInteger(    string  value      // string
   );
```

Parameters

value

[in] String containing a number.

Return Value

Value of long type.

StringToShortArray

The function symbol-wise copies a string into a specified place of an array of ushort type. It returns the number of copied elements.

```
int  StringToShortArray(    string  text_string,      // source string
   ushort& array[],           // array
   int     start=0,           // starting position in the array
   int     count=-1           // number of symbols
   );
```

Parameters

text_string

[in] String to copy

array[]

[out] Array of ushort type (analog of wchar_t type).

start=0

[in] Position, from which copying starts. Default - 0.

count=-1

[in] Number of array elements to copy. Defines length of a resulting string. Default value is -1, which means copying up to the array end, or till terminal 0.Terminal 0 will also be copied to the recipient array, in this case the size of a dynamic array can be increased if necessary to the size of the string. If the size of the dynamic array exceeds the length of the string, the size of the array will not be reduced.

Return Value

Number of copied elements.

StringToTime

The function converts a string containing time or date in "yyyy.mm.dd [hh:mi]" format into datetime type.

```
datetime  StringToTime(    string  value       // date string
   );
```

Parameters

value

[in] String in " yyyy.mm.dd hh:mi " format.

Return Value

Value of datetime type containing total number of seconds that elapsed since 01.01.1970.

StringFormat

The function formats obtained parameters and returns a string.

```
string  StringFormat(    string  format,      // string with format descrip
   ...      ...          // parameters
   );
```

Parameters

format

[in] String containing method of formatting. Formatting rules are the same as for the PrintFormat function.

...

[in] Parameters, separated by a comma.

Return Value

String.

Example:

```
void OnStart()
  {
//--- string variables
   string output_string;
   string temp_string;
   string format_string;
//--- prepare the specification header
   temp_string=StringFormat("Contract specification for %s:\r\n",_Symbol);
   StringAdd(output_string,temp_string);
//--- int value output
   int digits=(int)SymbolInfoInteger(_Symbol,SYMBOL_DIGITS);
   temp_string=StringFormat("   SYMBOL_DIGITS = %d (number of digits after
                            digits);
   StringAdd(output_string,temp_string);
//--- double value output with variable number of digits after the decimal
   double point_value=SymbolInfoDouble(_Symbol,SYMBOL_POINT);
   format_string=StringFormat("   SYMBOL_POINT = %%.%df (point value)\r\n"
                              digits);
   temp_string=StringFormat(format_string,point_value);
   StringAdd(output_string,temp_string);
//--- int value output
   int spread=(int)SymbolInfoInteger(_Symbol,SYMBOL_SPREAD);
   temp_string=StringFormat("   SYMBOL_SPREAD = %d (current spread in poin
                            spread);
   StringAdd(output_string,temp_string);
//--- int value output
   int min_stop=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
   temp_string=StringFormat("   SYMBOL_TRADE_STOPS_LEVEL = %d (minimal ind
                            min_stop);
   StringAdd(output_string,temp_string);
//--- double value output without the fractional part
   double contract_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_CONTRACT_SIZ
   temp_string=StringFormat("   SYMBOL_TRADE_CONTRACT_SIZE = %.f (contract
                            contract_size);
   StringAdd(output_string,temp_string);
//--- double value output with default accuracy
   double tick_size=SymbolInfoDouble(_Symbol,SYMBOL_TRADE_TICK_SIZE);
   temp_string=StringFormat("   SYMBOL_TRADE_TICK_SIZE = %f (minimal price
                            tick_size);
   StringAdd(output_string,temp_string);
//--- determining the swap calculation mode
   int swap_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_SWAP_MODE);
   string str_swap_mode;
   switch(swap_mode)
     {
      case 0: str_swap_mode="0 (in points)"; break;
      case 1: str_swap_mode="1 (in symbol base currency)"; break;
      case 2: str_swap_mode="2 (by interest)"; break;
      case 3: str_swap_mode="3 (in margin currency)"; break;
     }
//--- string value output
   temp_string=StringFormat("   SYMBOL_SWAP_MODE = %s\r\n",
                            str_swap_mode);
   StringAdd(output_string,temp_string);
//--- double value output with default accuracy
   double swap_long=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_LONG);
   temp_string=StringFormat("   SYMBOL_SWAP_LONG = %f (buy order swap valu
                            swap_long);
   StringAdd(output_string,temp_string);
//--- double value output with default accuracy
   double swap_short=SymbolInfoDouble(_Symbol,SYMBOL_SWAP_SHORT);
   temp_string=StringFormat("   SYMBOL_SWAP_SHORT = %f (sell order swap va
                            swap_short);
   StringAdd(output_string,temp_string);
//--- determining the trading mode
   int trade_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_MODE);
   string str_trade_mode;
   switch(trade_mode)
     {
      case SYMBOL_TRADE_MODE_DISABLED: str_trade_mode="SYMBOL_TRADE_MODE_DI
      case SYMBOL_TRADE_MODE_LONGONLY: str_trade_mode="SYMBOL_TRADE_MODE_LO
      case SYMBOL_TRADE_MODE_SHORTONLY: str_trade_mode="SYMBOL_TRADE_MODE_S
      case SYMBOL_TRADE_MODE_CLOSEONLY: str_trade_mode="SYMBOL_TRADE_MODE_C
      case SYMBOL_TRADE_MODE_FULL: str_trade_mode="SYMBOL_TRADE_MODE_FULL
     }
//--- string value output
   temp_string=StringFormat("   SYMBOL_TRADE_MODE = %s\r\n",
                            str_trade_mode);
   StringAdd(output_string,temp_string);
//--- double value output in a compact format
   double volume_min=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
   temp_string=StringFormat("   SYMBOL_VOLUME_MIN = %g (minimal volume for
   StringAdd(output_string,temp_string);
//--- double value output in a compact format
   double volume_step=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_STEP);
   temp_string=StringFormat("   SYMBOL_VOLUME_STEP = %g (minimal volume ch
   StringAdd(output_string,temp_string);
//--- double value output in a compact format
   double volume_max=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MAX);
   temp_string=StringFormat("   SYMBOL_VOLUME_MAX = %g (maximal volume for
   StringAdd(output_string,temp_string);
//--- determining the contract price calculation mode
   int calc_mode=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_CALC_MODE);
   string str_calc_mode;
   switch(calc_mode)
     {
      case 0:str_calc_mode="0 (Forex)";break;
      case 1:str_calc_mode="1 (CFD)";break;
      case 2:str_calc_mode="2 (futures)";break;
      case 3:str_calc_mode="3 (CFD for indices)";break;
     }
//--- string value output
   temp_string=StringFormat("   SYMBOL_TRADE_CALC_MODE = %s\r\n",
                            str_calc_mode);
   StringAdd(output_string,temp_string);
//--- double value output with 2 digits after the decimal point
   double margin_initial=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_INITIAL);
   temp_string=StringFormat("   SYMBOL_MARGIN_INITIAL = %.2f (initial marg
                            margin_initial);
   StringAdd(output_string,temp_string);
//--- double value output with 2 digits after the decimal point
   double margin_maintenance=SymbolInfoDouble(_Symbol,SYMBOL_MARGIN_MAINTE
   temp_string=StringFormat("   SYMBOL_MARGIN_MAINTENANCE = %.2f (maintena
                            margin_maintenance);
   StringAdd(output_string,temp_string);
//--- int value output
   int freeze_level=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_FREEZE_LEV
   temp_string=StringFormat("   SYMBOL_TRADE_FREEZE_LEVEL = %d (order free
                            freeze_level);
   StringAdd(output_string,temp_string);
   Print(output_string);
   Comment(output_string);
/* execution result
   Contract specification for EURJPY:
     SYMBOL_DIGITS = 3 (number of digits after the decimal point)
     SYMBOL_POINT = 0.001 (point value)
     SYMBOL_SPREAD = 23 (current spread in points)
     SYMBOL_TRADE_STOPS_LEVEL = 100 (minimal indention in points for Stop
     SYMBOL_TRADE_CONTRACT_SIZE = 100000 (contract size)
```

```
      SYMBOL_TRADE_TICK_SIZE = 0.001000 (minimal price change)
      SYMBOL_SWAP_MODE = 0 (in points)
      SYMBOL_SWAP_LONG = -1.600000 (buy order swap value)
      SYMBOL_SWAP_SHORT = -1.100000 (sell order swap value)
      SYMBOL_TRADE_MODE = SYMBOL_TRADE_MODE_FULL (no trade restrictions)
      SYMBOL_VOLUME_MIN = 0.01 (minimal volume for a deal)
      SYMBOL_VOLUME_STEP = 0.01 (minimal volume change step)
      SYMBOL_VOLUME_MAX = 1000 (maximal volume for a deal)
      SYMBOL_TRADE_CALC_MODE = 0 (Forex)
      SYMBOL_MARGIN_INITIAL = 0.00 (initial margin)
      SYMBOL_MARGIN_MAINTENANCE = 0.00 (maintenance margin)
      SYMBOL_TRADE_FREEZE_LEVEL = 20 (order freeze level in points)
*/
  }
```

See also

PrintFormat, DoubleToString,ColorToString, TimeToString

CharToStr

Conversion of the symbol code into a one-character string.

```
string  CharToStr(    uchar  char_code      // ASCII-code
   );
```

Parameters

char_code

[in] ASCII char code.

Returned value

Text string.

Example:

```
  string str="WORL" + CharToStr(68); // 68 is code for 'D'
  // the resulting string will be WORLD
```

See also

PrintFormat(), DoubleToString(), ColorToString(), TimeToString()

DoubleToStr

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

```
string  DoubleToStr(   double  value,     // value
   int     digits     // precision
   );
```

Parameters

value

[in] Floating point value.

digits

[in] Precision format, number of digits after decimal point (0-8).

Returned value

Text string.

Example:

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

See also

PrintFormat(), StrToDouble(), DoubleToString()

StrToDouble

Converts string representation of number to double type (double-precision format with floating point).

```
double  StrToDouble(    string  value       // value
   );
```

Parameters

value

[in] String containing the number character representation format.

Returned value

Value of double type.

Example:

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

See also

PrintFormat(), DoubleToStr(), DoubleToString()

StrToInteger

Converts string containing the value character representation into a value of the int (integer) type.

```
int  StrToInteger(    string  value       // string
   );
```

Parameters

value

[in] String containing the integer character representation format.

Returned value

Value of int type.

Example:

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

See also

PrintFormat(), StringToInteger(), IntegerToString()

StrToTime

Converts string in the format "yyyy.mm.dd hh:mi" to datetime type (the amount of seconds that have passed since 1 Jan., 1970).

```
datetime  StrToTime(    string  value       // string
   );
```

Parameters

value

[in] String having "yyyy.mm.dd hh:mi " format.

Returned value

Value of datetime type as a number of seconds, passed since 01.01.1970.

Example:

```
  datetime var1,var2,var3;
  var1=StrToTime("2003.8.12 17:35");
  var2=StrToTime("17:35");      // returns the current date with the given
  var3=StrToTime("2003.8.12");  // returns the date with the midnight time
```

See also

PrintFormat(), TimeToStr(), TimeToString()

TimeToStr

Converts value containing time in seconds that has passed since January 1, 1970, into a string of "yyyy.mm.dd hh:mi" format.

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

Parameters

value

[in] Positive amount of seconds that have passed since 00:00, January 1, 1970.

mode=TIME_DATE|TIME_MINUTES

[in] Optional data output mode can be one or combination of:
TIME_DATE gets result as "yyyy.mm.dd",
TIME_MINUTES gets result as "hh:mi",
TIME_SECONDS gets result as "hh:mi:ss".

Returned value

String.

Example:

```
   string var1=TimeToStr(TimeCurrent(),TIME_DATE|TIME_SECONDS);
```

See also

PrintFormat(), StrToTime(), StringToTime()

Mathematical Functions

A set of mathematical and trigonometric functions.

Function	Action
MathAbs	Returns absolute value (modulus) of the specified numeric value
MathArccos	Returns the arc cosine of x in radians
MathArcsin	Returns the arc sine of x in radians
MathArctan	Returns the arc tangent of x in radians
MathCeil	Returns integer numeric value closest from above
MathCos	Returns the cosine of a number
MathExp	Returns exponent of a number
MathFloor	Returns integer numeric value closest from below
MathLog	Returns natural logarithm
MathLog10	Returns the logarithm of a number by base 10
MathMax	Returns the maximal value of the two numeric values
MathMin	Returns the minimal value of the two numeric values
MathMod	Returns the real remainder after the division of two numbers
MathPow	Raises the base to the specified power
MathRand	Returns a pseudorandom value within the range of 0 to 32767
MathRound	Rounds of a value to the nearest integer
MathSin	Returns the sine of a number
MathSqrt	Returns a square root
MathSrand	Sets the starting point for generating a series of pseudorandom integers
MathTan	Returns the tangent of a number
MathIsValidNumber	Checks the correctness of a real number

MathAbs

The function returns the absolute value (modulus) of the specified numeric value.

```
double  MathAbs(    double  value        // numeric value
   );
```

Parameters

value

[in] Numeric value.

Return Value

Value of double type more than or equal to zero.

Note

Instead the MathAbs() function you can use fabs().

MathArccos

The function returns the arccosine of x within the range 0 to π in radians.

```
double  MathArccos(    double  val      // -1<val<1
   );
```

Parameters

val

[in] The val value between -1 and 1, the arc cosine of which is to be calculated.

Return Value

Arc cosine of a number in radians. If val is less than -1 or more than 1, the function returns NaN (indeterminate value).

Note

Instead of the MathArccos() function you can use acos().

See also

Real types (double, float)

MathArcsin

The function returns the arc sine of x within the range of -π/2 to π/2 radians.

```
double  MathArcsin(    double   val       // -1<value<1
   );
```

Parameters

val

[in] The val value between -1 and 1, the arc sine of which is to be calculated.

Return Value

Arc sine of the val number in radians within the range of -π/2 to π/2 radians. If val is less than -1 or more than 1, the function returns NaN (indeterminate value).

Note

Instead of the MathArcsin() function you can use asin().

See also

Real types (double, float)

MathArctan

The function returns the arc tangent of x. If x is equal to 0, the function returns 0.

```
double  MathArctan(    double   value      // tangent
   );
```

Parameters

value

[in] A number representing a tangent.

Return Value

MathArctan returns a value within the range of -π/2 to π/2 radians.

Note

Instead of the MathArctan() function you can use atan().

MathCeil

The function returns integer numeric value closest from above.

```
double  MathCeil(    double  val       // number
   );
```

Parameters

val

[in] Numeric value.

Return Value

Numeric value representing the smallest integer that exceeds or equals to val.

Note

Instead of the MathCeil() function you can use ceil().

MathCos

The function returns the cosine of an angle.

```
double  MathCos(    double  value       // number
   );
```

Parameters

value

[in] Angle in radians.

Return Value

Value of double type within the range of -1 to 1.

Note

Instead of MathCos() you can use cos().

MathExp

The function returns the value of e raised to the power of d.

```
double  MathExp(    double  value       // power for the number e
   );
```

Parameters

value

[in] A number specifying the power.

Return Value

A number of double type. In case of overflow the function returns INF (infinity), in case of underflow MathExp returns 0.

Note

Instead of MathExp() you can use exp().

See also

Real types (double, float)

MathFloor

The function returns integer numeric value closest from below.

```
double  MathFloor(    double  val      // number
   );
```

Parameters

val

[in] Numeric value.

Return Value

A numeric value representing the largest integer that is less than or equal to val.

Note

Instead of MathFloor() you can use floor().

MathLog

The function returns a natural logarithm.

```
double  MathLog(    double  val        // value to take the logarithm
   );
```

Parameters

val

[in] Value logarithm of which is to be found.

Return Value

The natural logarithm of val in case of success. If val is negative, the function returns NaN (undetermined value). If val is equal to 0, the function returns INF (infinity).

Note

Instead of MathLog() you can use log().

See also

Real types (double, float)

MathLog

Returns the logarithm of a number by base 10.

```
double  MathLog10(    double  val       // number to take logarithm
   );
```

Parameters

val

[in] Numeric value the common logarithm of which is to be calculated.

Return Value

The common logarithm in case of success. If val is negative, the function returns NaN (undetermined value). If val is equal to 0, the function returns INF (infinity).

Note

Instead of MathLog10() you can use log10().

See also

Real types (double, float)

MathMax

The function returns the maximal value of two values.

```
double  MathMax(    double  value1,     // first value
   double  value2      // second value
   );
```

Parameters

value1

[in] First numeric value.

value2

[in] Second numeric value.

Return Value

The largest of the two values.

Note

Instead of MathMax() you can use fmax(). Functions fmax(), fmin(), MathMax(), MathMin() can work with integer types without typecasting them to the type of double.

If parameters of different types are passed into a function, the parameter of the smaller type is automatically cast to the larger type. The type of the return value corresponds to the larger type.

If data of the same type are passed, no casting is performed.

MathMin

The function returns the minimal value of two values.

```
double  MathMin(    double  value1,      // first value
   double   value2       // second value
   );
```

Parameters

value1

[in] First numeric value.

value2

[in] Second numeric value.

Return Value

The smallest of the two values.

Note

Instead of MathMin() you can use fmin(). Functions fmax(), fmin(), MathMax(), MathMin() can work with integer types without typecasting them to the type of double.

If parameters of different types are passed into a function, the parameter of the smaller type is automatically cast to the larger type. The type of the return value corresponds to the larger type.

If data of the same type are passed, no casting is performed.

MathMod

The function returns the real remainder of division of two numbers.

```
double  MathMod(    double   value,       // dividend value
   double   value2       // divisor value
   );
```

Parameters

value

[in] Dividend value.

value2

[in] Divisor value.

Return Value

The MathMod function calculates the real remainder f from expression val/y so that val = i * y + f , where i is an integer, f has the same sign as val, and the absolute value of f is less than the absolute value of y.

Note

Instead of MathMod() you can use fmod().

MathPow

The function raises a base to a specified power.

```
double  MathPow(    double  base,          // base
   double  exponent      // exponent value
   );
```

Parameters

base

[in] Base.

exponent

[in] Exponent value.

Return Value

Value of base raised to the specified power.

Note

Instead of MathPow() you can use pow().

MathRand

Returns a pseudorandom integer within the range of 0 to 32767.

```
int  MathRand();
```

Return Value

Integer value within the range of 0 to 32767.

Note

Before the first call of the function, it's necessary to call MathSrand to set the generator of pseudorandom numbers to the initial state.

Note

Instead of MathRand() you can use rand().

MathRound

The function returns a value rounded off to the nearest integer of the specified numeric value.

```
double  MathRound(    double  value       // value to be rounded
   );
```

Parameters

value

[in] Numeric value before rounding.

Return Value

Value rounded till to the nearest integer.

Note

Instead of MathRound() you can use round().

MathSin

Returns the sine of a specified angle.

```
double  MathSin(    double  value       // argument in radians
   );
```

Parameters

value

[in] Angle in radians.

Return Value

Sine of an angle measured in radians. Returns value within the range of -1 to 1.

Note

Instead of MathSin() you can use sin().

MathSqrt

Returns the square root of a number.

```
double  MathSqrt(    double  value      // positive number
   );
```

Parameters

value

[in] Positive numeric value.

Return Value

Square root of value. If value is negative, MathSqrt returns NaN (indeterminate value).

Note

Instead of MathSqrt() you can use sqrt().

See also

Real types (double, float)

MathSrand

Sets the starting point for generating a series of pseudorandom integers.

```
void  MathSrand(    int  seed       // initializing number
   );
```

Parameters

seed

[in] Starting number for the sequence of random numbers.

Return Value

No return value.

Note

The MathRand() function is used for generating a sequence of pseudorandom numbers. Call of MathSrand() with a certain initializing number allows to always produces the same sequence of pseudorandom numbers.

To ensure receipt of non-recurring sequence, use the call of MathSrand(GetTickCount()), since the value of GetTickCount() increases from the moment of the start of the operating system and is not repeated within 49 days, until the built-in counter of milliseconds overflows. Use of MathSrand(TimeCurrent()) is not suitable, because the TimeCurrent() function returns the time of the last tick, which can be unchanged for a long time, for example at the weekend.

Initialization of the random number generator using MathSrand() for indicators and Expert Advisors is better performed in the OnInit() handler; it saves you from the following multiple restarts of the generator in OnTick() and OnCalculate().

Instead of the MathSrand() function you can use the srand() function.

Example:

```
#property description "The indicator shows the central limit theorem, whic
#property description "The sum of a sufficiently large number of weakly de
#property description "having approximately equal magnitude (none of the s
#property description "or makes a determining contribution to the sum), ha

#property indicator_separate_window
#property indicator_buffers 1
//--- Properties of the graphical construction
#property indicator_label1  "Label"
```

```
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRoyalBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  5
//--- An input variable
input int      sample_number=10;
//--- An indicator buffer to for drawing the distribution
double         LabelBuffer[];
//--- A counter of ticks
double         ticks_counter;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- Binding an array and an indicator buffer
   SetIndexBuffer(0,LabelBuffer,INDICATOR_DATA);
//--- turn the indicator buffer around from the present to the past
   ArraySetAsSeries(LabelBuffer,true);
//--- Initialize the generator of random numbers
   MathSrand(GetTickCount());
//--- Initialize the counter of ticks
   ticks_counter=0;
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- For a zero counter reset the indicator buffer
   if(ticks_counter==0) ArrayInitialize(LabelBuffer,0);
//--- Increase the counter
   ticks_counter++;
//--- We should periodically reset the counter ticks, to revive the distri
   if(ticks_counter>100)
     {
      Print("We've reset the indicator values, let's start filling the cel
      ticks_counter=0;
     }
//--- Get a sample of random values as the sum of three numbers from 0 to
   for(int i=0;i<sample_number;i++)
     {
      //--- Calculation of the index of the cell, where the random number
      int rand_index=0;
      //--- Get three random numbers from 0 to 7
      for(int k=0;k<3;k++)
        {
         //--- A remainder in the division by 7 will return a value from
         rand_index+=MathRand()%7;
        }
      //--- Increase the value in the cell number rand_index by 1
      LabelBuffer[rand_index]++;
     }
//--- Exit the OnCalculate() handler
   return(rates_total);
  }
```

MathTan

The function returns a tangent of a number.

```
double  MathTan(    double  rad       // argument in radians
   );
```

Parameters

rad

[in] Angle in radians.

Return Value

Tangent of rad. If rad 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 number.

Note

Instead of MathTan() you can use tan().

See also

Real types (double, float)

MathIsValidNumber

It checks the correctness of a real number.

```
bool  MathIsValidNumber(    double   number       // number to check
   );
```

Parameters

number

[in] Checked numeric value.

Return Value

It returns true, if the checked value is an acceptable real number. If the checked value is a plus or minus infinity, or "not a number" (NaN), the function returns false.

Example:

```
   double abnormal=MathArcsin(2.0);
   if(!MathIsValidNumber(abnormal)) Print("Attention! MathArcsin(2.0) = ",
```

See also

[Real types (double, float)]

String Functions

This is a group of functions intended for working with data of the string type.

Function	Action
StringAdd	Adds a string to the end of another string
StringBufferLen	Returns the size of buffer allocated for the string
StringCompare	Compares two strings and returns 1 if the first string is greater than the second; 0 - if the strings are equal; -1 (minus 1) - if the first string is less than the second one
StringConcatenate	Forms a string of parameters passed
StringFill	Fills out a specified string by selected symbols
StringFind	Search for a substring in a string
StringGetCharacter	Returns the value of a number located in the specified string position
StringInit	Initializes string by specified symbols and provides the specified string length
StringLen	Returns the number of symbols in a string
StringReplace	Replaces all the found substrings of a string by a set sequence of symbols
StringSetCharacter	Returns a copy of a string with a changed value of a symbol in a specified position
StringSplit	Gets substrings by a specified separator from the specified string, returns the number of substrings obtained
StringSubstr	Extracts a substring from a text string starting from a specified position
StringToLower	Transforms all symbols of a selected string to lowercase
StringToUpper	Transforms all symbols of a selected string into capitals
StringTrimLeft	Cuts line feed characters, spaces and tabs in the left part of the string
StringTrimRight	Cuts line feed characters, spaces and tabs in the right part of the string
StringGetChar	Returns character (code) from the specified position in the string

StringSetChar	Returns the string copy with changed character in the specified position

StringAdd

The function adds a substring to the end of a string.

```
bool  StringAdd(    string&  string_var,         // string, to which we add
   string    add_substring        // string, which is added
   );
```

Parameters

string_var

[in][out] String, to which another one is added.

add_substring

[in] String that is added to the end of a source string.

Return Value

In case of success returns true, otherwise false. In order to get an error code, the GetLastError() function should be called.

Example:

```
void OnStart()
  {
//---
   long length=10000000;
   string a="a",b="b",c;
//--- first method
   uint starttime=GetTickCount(),finishtime;
   long i;
   for(i=0;i<length;i++)
     {
      c=a+b;
     }
   finishtime=GetTickCount();
   Print("time for 'c = a + b' = ",(finishtime-starttime)," milliseconds,

//--- second method
   starttime=GetTickCount();
   for(i=0;i<length;i++)
     {
      StringAdd(a,b);
     }
   finishtime=GetTickCount();
   Print("time for 'StringAdd(a,b)' = ",(finishtime-starttime)," millisec

//--- third method
   starttime=GetTickCount();
   a="a"; //  re-initialize variable a
   for(i=0;i<length;i++)
     {
      c=StringConcatenate(a,b);
     }
   finishtime=GetTickCount();
   Print("time for 'c=StringConcatenate(a,b)' = ",(finishtime-starttime),"
  }
```

See also

StringConcatenate()

StringBufferLen

The function returns the size of buffer allocated for the string.

```
int  StringBufferLen(    string  string_var      // string
   )
```

Parameters

string_var

[in] String.

Return Value

The value 0 means that the string is constant and buffer size can't be changed. -1 means that the string belongs to the client terminal, and modification of the buffer contents can have indeterminate results.

Note

Minimal buffer size is equal to 16.

Example:

```
void OnStart()
  {
   long length=1000;
   string a="a",b="b";
//---
   long i;
   Print("before: StringBufferLen(a) = ",StringBufferLen(a),
         "  StringLen(a) = ",StringLen(a));
   for(i=0;i<length;i++)
     {
      StringAdd(a,b);
     }
   Print("after: StringBufferLen(a) = ",StringBufferLen(a),
         "  StringLen(a) = ",StringLen(a));
  }
```

See also

StringAdd(), StringInit(), StringLen(), StringFill()

StringCompare

The function compares two strings and returns the comparison result in form of an integer.

```
int  StringCompare(    const string&  string1,                // the firs
   const string&  string2,                 // the second string in the com
   bool           case_sensitive=true      // case sensitivity mode select
   );
```

Parameters

string1

[in] The first string.

string2

[in] The second string.

case_sensitive=true

[in] Case sensitivity mode selection. If it is true, then "A">"a". If it is false, then "A"="a". By default the value is equal to true.

Return Value

- -1 (minus one), if string1<string2
- 0 (zero), if string1=string2
- 1 (one), if string1>string2

Note

The strings are compared symbol by symbol, the symbols are compared in the alphabetic order in accordance with the current code page.

Example:

```
void OnStart()
  {
//--- what is larger - apple or home?
   string s1="Apple";
   string s2="home";

//--- compare case sensitive
   int result1=StringCompare(s1,s2);
   if(result1>0) PrintFormat("Case sensitive comparison: %s > %s",s1,s2);
   else
     {
      if(result1<0)PrintFormat("Case sensitive comparison: %s < %s",s1,s2)
      else PrintFormat("Case sensitive comparison: %s = %s",s1,s2);
     }

//--- compare case-insensitive
   int result2=StringCompare(s1,s2,false);
   if(result2>0) PrintFormat("Case insensitive comparison: %s > %s",s1,s2)
   else
     {
      if(result2<0)PrintFormat("Case insensitive comparison: %s < %s",s1,s
      else PrintFormat("Case insensitive comparison: %s = %s",s1,s2);
     }
/* Result:
     Case-sensitive comparison: Apple < home
     Case insensitive comparison: Apple < home
*/
  }
```

See also

String Type, CharToString(), ShortToString(), StringToCharArray(), StringToShortArray(), StringGetCharacter(), Use of a Codepage

StringConcatenate

The function returns the string formed by concatenation of parameters transformed into string type.

```
string  StringConcatenate(    void argument1,        // first parameter of
   void argument2,          // second parameter of any simple type
   ...                      // next parameter of any simple type
   );
```

Parameters

argumentN

[in] Any comma separated values. From 2 to 63 parameters of any simple type.

Return Value

Returns the string, formed by concatenation of parameters transformed into string type. Parameters are transformed into strings according to the same rules as in Print() and Comment().

Note

Parameters can be of any type. Number of parameters can't be less than 2 or more than 64.

Example:

```
   string text;
   text=StringConcatenate("Account free margin is ", AccountFreeMargin(), "
   // text="Account free margin is " + AccountFreeMargin() + "  Current tim
   Print(text);
```

See also

StringAdd()

StringFill

It fills out a selected string by specified symbols.

```
bool  StringFill(    string&    string_var,        // string to fill
   ushort     character          // symbol that will fill the string
   );
```

Parameters

string_var

[in][out] String, that will be filled out by the selected symbol.

character

[in] Symbol, by which the string will be filled out.

Return Value

In case of success returns true, otherwise - false. To get the error code call GetLastError().

Note

Filling out a string at place means that symbols are inserted directly to the string without transitional operations of new string creation or copying. This allows to save the operation time.

Example:

```
void OnStart()
  {
   string str;
   StringInit(str,20,'_');
   Print("str = ",str);
   StringFill(str,0);
   Print("str = ",str,": StringBufferLen(str) = ", StringBufferLen(str));
  }
// Result
//    str = ____________________
//    str =   : StringBufferLen(str) = 20
//
```

See also

StringBufferLen(), StringLen(), StringInit()

StringFind

Search for a substring in a string.

```
int  StringFind(    string  string_value,        // string in which search
   string  match_substring,     // what is searched
   int     start_pos=0          // from what position search starts
   );
```

Parameters

string_value

[in] String, in which search is made.

match_substring

[in] Searched substring.

start_pos=0

[in] Position in the string from which search is started.

Return Value

Returns position number in a string, from which the searched substring starts, or -1, if the substring is not found.

StringGetCharacter

Returns value of a symbol, located in the specified position of a string.

```
ushort  StringGetCharacter(    string  string_value,     // string
   int     pos               // symbol position in the string
   );
```

Parameters

string_value

[in] String.

pos

[in] Position of a symbol in the string. Can be from 0 to StringLen(text) -1.

Return Value

Symbol code or 0 in case of an error. To get the error code call GetLastError().

StringInit

Initializes a string by specified symbols and provides the specified string size.

```
bool  StringInit(   string&   string_var,       // string to initialize
   int       new_len=0,        // required string length after initializat
   ushort    character=0       // symbol, by which the string will be fill
   );
```

Parameters

string_var

[in][out] String that should be initialized and deinitialized.

new_len=0

[in] String length after initialization. If length=0, it deinitializes the string, i.e. the string buffer is cleared and the buffer address is zeroed.

character=0

[in] Symbol to fill the string.

Return Value

In case of success returns true, otherwise - false. To get the error code call GetLastError().

Note

If character=0 and the length new_len>0, the buffer of the string of indicated length will be distributed and filled by zeroes. The string length will be equal to zero, because the whole buffer is filled out by string terminators.

Example:

```
void OnStart()
  {
//---
   string str;
   StringInit(str,200,0);
   Print("str = ",str,": StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
  }
/*  Result
str = : StringBufferLen(str) = 200   StringLen(str) = 0
*/
```

See also

StringBufferLen(), StringLen()

StringLen

Returns the number of symbols in a string.

```
int  StringLen(    string   string_value       // string
   );
```

Parameters

string_value

[in] String to calculate length.

Return Value

Number of symbols in a string without the ending zero.

StringReplace

It replaces all the found substrings of a string by a set sequence of symbols.

```
int  StringReplace(    string&          str,              // the string in
   const string     find,             // the searched substring
   const string     replacement       // the substring that will be inserte
   );
```

Parameters

str

[in][out] The string in which you are going to replace substrings.

find

[in] The desired substring to replace.

replacement

[in] The string that will be inserted instead of the found one.

Return Value

The function returns the number of replacements in case of success, otherwise -1. To get an error code call the GetLastError() function.

Note

If the function has run successfully but no replacements have been made (the substring to replace was not found), it returns 0.

The error can result from incorrect *str* or *find* parameters (empty or non-initialized string, see StringInit()). Besides, the error occurs if there is not enough memory to complete the replacement.

Example:

```
   string text="The quick brown fox jumped over the lazy dog.";
   int replaced=StringReplace(text,"quick","slow");
   replaced+=StringReplace(text,"brown","black");
   replaced+=StringReplace(text,"fox","bear");
   Print("Replaced: ", replaced,". Result=",text);

//  Result
//  Replaced: 3. Result=The slow black bear jumped over the lazy dog.
//
```

See also

StringSetCharacter(), StringSubstr()

StringSetCharacter

Returns true is a symbol is successfully inserted to the passed string.

```
bool  StringSetCharacter(    string&    string_var,        // string
   int        pos,                // position
   ushort     character           // character
   );
```

Parameters

string_var

[in][out] String.

pos

[in] Position of a character in a string. Can be from 0 to StringLen(text).

character

[in] Symbol code Unicode.

Return Value

In case of success returns true, otherwise - false.

Note

If pos is less than string length and the symbol code value = 0, the string is cut off (but the buffer size, distributed for the string remains unchanged). The string length becomes equal to pos.

If *pos* is equal to string length, the specified symbol is added at the string end, and the length is enlarged by one.

Example:

```
void OnStart()
  {
   string str="0123456789";
   Print("before: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
//--- add zero value in the middle
   StringSetCharacter(str,6,0);
   Print("after: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
//--- add symbol at the end
   int size=StringLen(str);
   StringSetCharacter(str,size,'+');
   Print("addition: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
  }
/* Result
   before: str = 0123456789 ,StringBufferLen(str) = 0   StringLen(str) = 1
    after: str = 012345 ,StringBufferLen(str) = 16   StringLen(str) = 6
   addition: str = 012345+ ,StringBufferLen(str) = 16   StringLen(str) = 7
*/
```

See also

StringBufferLen(), StringLen(), StringFill(), StringInit()

StringSplit

Gets substrings by a specified separator from the specified string, returns the number of substrings obtained.

```
int  StringSplit(    const string    string_value,       // A string to sea
   const ushort    separator,           // A separator using which substring
   string          & result[]           // An array passed by reference to g
   );
```

Parameters

string_value

[in] The string from which you need to get substrings. The string will not change.

pos

[in] The code of the separator character. To get the code, you can use the StringGetCharacter() function.

result[]

[out] An array of strings where the obtained substrings are located.

Return Value

The number of substrings in the result[] array. If the separator is not found in the passed string, only one source string will be placed in the array.

If string_value is empty or NULL, the function will return zero. In case of an error the function returns -1. To get the error code, call the GetLastError() function.

Example:

```
string to_split="life_is_good";   // A string to split into substrings
   string sep="_";                // A separator as a character
   ushort u_sep;                  // The code of the separator character
   string result[];               // An array to get strings
   //--- Get the separator code
   u_sep=StringGetCharacter(sep,0);
   //--- Split the string to substrings
   int k=StringSplit(to_split,u_sep,result);
   //--- Show a comment
   PrintFormat("Strings obtained: %d. Used separator '%s' with the code %d
   //--- Now output all obtained strings
   if(k>0)
     {
      for(int i=0;i<k;i++)
        {
         PrintFormat("result[%d]=%s",i,result[i]);
        }
     }
```

See also

StringReplace(), StringSubstr(), StringConcatenate()

StringSubstr

Extracts a substring from a text string starting from the specified position.

```
string  StringSubstr(    string  string_value,      // string
   int      start_pos,        // position to start with
   int      length=0          // length of extracted string
   );
```

Parameters

string_value

[in] String to extract a substring from.

start_pos

[in] Initial position of a substring. Can be from 0 to StringLen(text) -1.

length=0

[in] Length of an extracted substring. If the parameter value is equal or less than 0 or parameter isn't set, the substring will be extracted from the indicated position till the string end.

Return Value

Copy of a extracted substring, if possible. Otherwise returns an empty string.

StringToLower

Transforms all symbols of a selected string into lowercase.

```
bool  StringToLower(    string&   string_var        // string to process
   );
```

Parameters

string_var

[in][out] String.

Return Value

In case of success returns true, otherwise - false. To get the error code call GetLastError().

StringToUpper

Transforms all symbols of a selected string into capitals.

```
bool  StringToUpper(    string&  string_var      // string to process
   );
```

Parameters

string_var

[in][out] String.

Return Value

In case of success returns true, otherwise - false. To get the error code call GetLastError().

StringTrimLeft

The function cuts line feed characters, spaces and tabs in the left part of the string till the first meaningful symbol. The string is modified at place.

```
string  StringTrimLeft(    const string  text       // string to cut
   );
```

Parameters

text

[in] String that will be cut from the left.

Return Value

A copy of the cut string if possible, otherwise an empty string.

Example:

```
  string str1="  Hello world   ";
  string str2=StringTrimLeft(str1);
  // after left trim str2 will be equal to "Hello World   "
```

StringTrimRight

The function cuts line feed characters, spaces and tabs in the right part of the string after the last meaningful symbol. The string is modified at place.

```
string  StringTrimRight(    const string   text       // string to cut
   );
```

Parameters

text

[in] String that will be cut from the right.

Return Value

A copy of the cut string if possible, otherwise an empty string.

Exampe:

```
  string str1 = "  Hello world    ";
  string str2=StringTrimRight(str1);
  // after right trim str2 will be equal to "  Hello World"
```

StringGetChar

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

```
ushort  StringGetChar(    string  string_value,     // string
   int     pos                    // position
   );
```

Parameters

string_value

[in] String.

pos

[in] Char position in the string. Can be from 0 to StringLen(text) -1.

Returned value

Char code or 0 if error. To get information about error, call the GetLastError() function.

Example:

```
  int char_code=StringGetChar("abcdefgh", 2);
  // char code of 'c' = 99
```

StringSetChar

Returns the string copy with changed character in the specified position.

```
string  StringSetChar(    string     string_var,        // string
   int       pos,                 // position
   ushort    value                // char code
   );
```

Parameters

string_var

[in] Source string.

pos

[in] The character position in the string. Can be from 0 to StringLen(text).

value

[in] New char ASCII-code.

Returned value

The string copy with changed character in the specified position.

Note

If the value of pos parameter is less than string length and char code = 0, the string will be truncated (to position, equal to pos). If the value of *pos* parameter is equal to string lenth, the specified symbol will be added to the end of the string and string length will be increased by 1.

Example:

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

See also

StringBufferLen(), StringLen(), StringFill(), StringInit()

Date and Time

This is the group of functions for working with data of datetime type (an integer that represents the number of seconds elapsed from 0 hours of January 1, 1970).

To arrange high-resolution counters and timers, use the GetTickCount() function, which produces values in milliseconds.

Function	Action
TimeCurrent	Returns the last known server time (time of the last quote receipt) in the datetime format
TimeLocal	Returns the local computer time in datetime format
TimeGMT	Returns GMT in datetime format with the Daylight Saving Time by local time of the computer, where the client terminal is running
TimeDaylightSavings	Returns the sign of Daylight Saving Time switch
TimeGMTOffset	Returns the current difference between GMT time and the local computer time in seconds, taking into account DST switch
TimeToStruct	Converts a datetime value into a variable of MqlDateTime structure type
StructToTime	Converts a variable of MqlDateTime structure type into a datetime value
Day	Returns the current day of the month, i.e., the day of month of the last known server time
DayOfWeek	Returns the current zero-based day of the week of the last known server time
DayOfYear	Returns the current day of the year i.e., the day of year of the last known server time
Hour	Returns the hour of the last known server time by the moment of the program start
Minute	Returns the current minute of the last known server time by the moment of the program start
Month	Returns the current month as number, i.e., the number of month of the last known server time
Seconds	Returns the amount of seconds elapsed from the beginning of the current minute of the last known server time by the moment of

	the program start
TimeDay	Returns the day of month of the specified date
TimeDayOfWeek	Returns the zero-based day of week of the specified date
TimeDayOfYear	Returns the day of year of the specified date
TimeHour	Returns the hour of the specified time
TimeMinute	Returns the minute of the specified time
TimeMonth	Returns the month number of the specified time
TimeSeconds	Returns the amount of seconds elapsed from the beginning of the minute of the specified time
TimeYear	Returns year of the specified date
Year	Returns the current year, i.e., the year of the last known server time

TimeCurrent

Returns the last known server time, time of the last quote receipt for one of the symbols selected in the "Market Watch" window. In the OnTick() handler, this function returns the time of the received handled tick. In other cases (for example, call in handlers OnInit(), OnDeinit(), OnTimer() and so on) this is the time of the last quote receipt for any symbol available in the "Market Watch" window, the time shown in the title of this window. The time value is formed on a trade server and does not depend on the time settings on your computer. There are 2 variants of the function.

Call without parameters

```
datetime  TimeCurrent();
```

Call with MqlDateTime type parameter

```
datetime  TimeCurrent(    MqlDateTime&  dt_struct      // structure type v
   );
```

Parameters

dt_struct

[out] MqlDateTime structure type variable.

Return Value

Value of datetime type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the GetTickCount() function, which produces values in milliseconds.

During testing in the Strategy Tester, TimeCurrent() is simulated according to historical data.

TimeLocal

Returns the local time of a computer, where the client terminal is running. There are 2 variants of the function.

Call without parameters

```
datetime  TimeLocal();
```

Call with MqlDateTime type parameter

```
datetime  TimeLocal(    MqlDateTime&  dt_struct       // Variable of struct
   );
```

Parameters

dt_struct

[out] Variable of structure type MqlDateTime.

Return Value

Value of datetime type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the GetTickCount() function, which produces values in milliseconds.

During testing in the Strategy Tester, TimeLocal() is always equal to TimeCurrent() simulated server time.

TimeGMT

Returns the GMT, which is calculated taking into account the DST switch by the local time on the computer where the client terminal is running. There are 2 variants of the function.

Call without parameters

```
datetime  TimeGMT();
```

Call with MqlDateTime type parameter

```
datetime  TimeGMT(    MqlDateTime&  dt_struct      // Variable of structure
   );
```

Parameters

dt_struct

[out] Variable of structure type MqlDateTime.

Return Value

Value of datetime type

Note

If the MqlDateTime structure type variable has been passed as a parameter, it is filled accordingly.

To arrange high-resolution counters and timers, use the GetTickCount() function, which produces values in milliseconds.

During testing in the Strategy Tester, TimeGMT() is always equal to TimeCurrent() simulated server time.

TimeDaylightSavings

Returns correction for daylight saving time in seconds, if the switch to summer time has been made. It depends on the time settings of your computer.

```
int  TimeDaylightSavings();
```

Return Value

If switch to winter (standard) time has been made, it returns 0.

TimeGMTOffset

Returns the current difference between GMT time and the local computer time in seconds, taking into account switch to winter or summer time. Depends on the time settings of your computer.

```
int  TimeGMTOffset();
```

Return Value

The value of int type, representing the current difference between GMT time and the local time of the computer TimeLocal in seconds.

```
TimeGMTOffset() =  TimeGMT() - TimeLocal()
```

TimeToStruct

Converts a value of datetime type (number of seconds since 01.01.1970) into a structure variable MqlDateTime.

```
void  TimeToStruct(    datetime      dt,            // date and time
   MqlDateTime&  dt_struct      // structure for the adoption of values
   );
```

Parameters

dt

[in] Date value to convert.

dt_struct

[out] Variable of structure type MqlDateTime.

Return Value

No return value.

StructToTime

Converts a structure variable MqlDateTime into a value of datetime type and returns the resulting value.

```
datetime  StructToTime(    MqlDateTime$  dt_struct      // structure of th
   );
```

Parameters

dt_struct

[in] Variable of structure type MqlDateTime.

Return Value

The value of datetime type containing the number of seconds since 01.01.1970.

Day

Returns the current day of the month, i.e., the day of month of the last known server time.

```
int  Day();
```

Returned value

Current day of the month.

Note

At the testing, the last known server time is modelled.

Example:

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

DayOfWeek

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

```
int  DayOfWeek();
```

Returned value

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

Note

At the testing, the last known server time is modelled.

Example:

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

DayOfYear

Returns the current day of the year (1 means 1 January,..,365(6) does 31 December), i.e., the day of year of the last known server time.

```
int  DayOfYear();
```

Returned value

Current day of the year (1 means 1 January,..,365(6) does 31 December).

Note

At the testing, the last known server time is modelled.

Example:

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

Hour

Returns the hour (0,1,2,..23) of the last known server time by the moment of the program start.

```
int  Hour();
```

Returned value

The hour (0,1,2,..23) of the last known server time by the moment of the program start (this value will not change within the time of the program execution).

Note

At the testing, the last known server time is modelled.

Example:

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

Minute

Returns the current minute (0,1,2,..59) of the last known server time by the moment of the program start.

```
int  Minute();
```

Returned value

Returns the current minute (0,1,2,..59) of the last known server time by the moment of the program start (this value will not change within the time of the program execution).

Note

At the testing, the last known server time is modelled.

Example:

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

Month

Returns the current month as number (1-January,2,3,4,5,6,7,8,9,10,11,12), i.e., the number of month of the last known server time.

```
int  Month();
```

Returned value

Returns the current month as number (1-January,2,3,4,5,6,7,8,9,10,11,12), i.e., the number of month of the last known server time.

Note

At the testing, the last known server time is modelled.

Example:

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

Seconds

Returns the amount of seconds elapsed from the beginning of the current minute of the last known server time by the moment of the program start.

```
int  Seconds();
```

Returned value

The amount of seconds elapsed from the beginning of the current minute of the last known server time by the moment of the program start (this value will not change within the time of the program execution).

Note

At the testing, the last known server time is modelled.

Example:

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

TimeDay

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

```
int  TimeDay(    datetime      date             // date and time
   );
```

Parameters

date

[in] Datetime as number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Returned value

Day of month (1 - 31) of the specified date.

Example:

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

TimeDayOfWeek

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

```
int  TimeDayOfWeek(    datetime      date              // date and time
   );
```

Parameters

date

[in] Datetime as number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Returned value

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

Example:

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

TimeDayOfYear

Returns the day of year of the specified date.

```
int  TimeDayOfYear(    datetime      date             // date and time
   );
```

Parameters

date

[in] Datetime as number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Returned value

Day (1 means 1 January,..,365(6) does 31 December) of year of the specified date.

Example:

```
  int nday=TimeDayOfYear(TimeCurrent());
```

TimeHour

Returns the hour of the specified time.

```
int  TimeHour(    datetime      date            // date and time
   );
```

Parameters

date

[in] Datetime is the number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Returned value

Hour of the specified time.

Example:

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

TimeMinute

Returns the minute of the specified time.

```
int  TimeMinute(    datetime      date              // date and time
   );
```

Parameters

date

[in] Datetime as number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Returned value

Minute (0-59) for the specified time.

Example:

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

TimeMonth

Returns the month number of the specified time.

```
int  TimeMonth(    datetime      date            // date and time
   );
```

Parameters

date

[in] Datetime as number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Returned value

Month number (1-January,2,3,4,5,6,7,8,9,10,11,12) of the specified time.

Example:

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

TimeSeconds

Returns the amount of seconds elapsed from the beginning of the minute of the specified time.

```
int  TimeSeconds(    datetime      date             // date and time
   );
```

Parameters

date

[in] Datetime as number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Returned value

The amount of seconds elapsed from the beginning of the minute of the specified time.

Example:

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

TimeYear

Returns year of the specified date.

```
int  TimeYear(    datetime      date              // date and time
   );
```

Parameters

date

[in] Datetime as number of seconds elapsed since midnight (00:00:00), January 1, 1970.

Returned value

Year of the specified date. The returned value can be within the range of 1970 to 3000.

Example:

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

Year

Returns the current year, i.e., the year of the last known server time.

```
int  Year();
```

Returned value

Current year.

Note

At the testing, the last known server time is modelled.

Example:

```
  // return if the date is within the range from 1 Jan. to 30 Apr., 2006.
    return(0);
```

Account Information

Functions that return parameters of the current account.

Function	Action
AccountInfoDouble	Returns a value of double type of the corresponding account property
AccountInfoInteger	Returns a value of integer type (bool, int or long) of the corresponding account property
AccountInfoString	Returns a value string type corresponding account property
AccountBalance	Returns balance value of the current account
AccountCredit	Returns credit value of the current account
AccountCompany	Returns the brokerage company name where the current account was registered
AccountCurrency	Returns currency name of the current account
AccountEquity	Returns equity value of the current account
AccountFreeMargin	Returns free margin value of the current account
AccountFreeMarginCheck	Returns free margin that remains after the specified position has been opened at the current price on the current account
AccountFreeMarginMode	Calculation mode of free margin allowed to open orders on the current account
AccountLeverage	Returns leverage of the current account
AccountMargin	Returns margin value of the current account
AccountName	Returns the current account name
AccountNumber	Returns the current account number
AccountProfit	Returns profit value of the current account
AccountServer	Returns the connected server name
AccountStopoutLevel	Returns the value of the Stop Out level
AccountStopoutMode	Returns the calculation mode for the Stop Out level

AccountInfoDouble

Returns the value of the corresponding account property.

```
double  AccountInfoDouble(    int  property_id      // identifier of the p
   );
```

Parameters

property_id

[in] Identifier of the property. The value can be one of the values of ENUM_ACCOUNT_INFO_DOUBLE.

Return Value

Value of double type.

Example:

```
void OnStart()
  {
//--- show all the information available from the function AccountInfoDoub
   printf("ACCOUNT_BALANCE =  %G",AccountInfoDouble(ACCOUNT_BALANCE));
   printf("ACCOUNT_CREDIT =  %G",AccountInfoDouble(ACCOUNT_CREDIT));
   printf("ACCOUNT_PROFIT =  %G",AccountInfoDouble(ACCOUNT_PROFIT));
   printf("ACCOUNT_EQUITY =  %G",AccountInfoDouble(ACCOUNT_EQUITY));
   printf("ACCOUNT_MARGIN =  %G",AccountInfoDouble(ACCOUNT_MARGIN));
   printf("ACCOUNT_MARGIN_FREE =  %G",AccountInfoDouble(ACCOUNT_FREEMARGIN
   printf("ACCOUNT_MARGIN_LEVEL =  %G",AccountInfoDouble(ACCOUNT_MARGIN_LE
   printf("ACCOUNT_MARGIN_SO_CALL = %G",AccountInfoDouble(ACCOUNT_MARGIN_S
   printf("ACCOUNT_MARGIN_SO_SO = %G",AccountInfoDouble(ACCOUNT_MARGIN_SO_
  }
```

See also

SymbolInfoDouble, SymbolInfoString, SymbolInfoInteger, PrintFormat

AccountInfoInteger

Returns the value of the properties of the account.

```
long  AccountInfoInteger(    int  property_id      // Identifier of the pr
   );
```

Parameters

property_id

[in] Identifier of the property. The value can be one of the values of ENUM_ACCOUNT_INFO_INTEGER.

Return Value

Value of long type.

Note

The property must be one of the bool, int or long types.

Example:

```
void OnStart()
  {
//--- show all the information available from the function AccountInfoInte
   printf("ACCOUNT_LOGIN =  %d",AccountInfoInteger(ACCOUNT_LOGIN));
   printf("ACCOUNT_LEVERAGE =  %d",AccountInfoInteger(ACCOUNT_LEVERAGE));
   bool thisAccountTradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_ALLOWED);
   bool EATradeAllowed=AccountInfoInteger(ACCOUNT_TRADE_EXPERT);
   ENUM_ACCOUNT_TRADE_MODE tradeMode=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoI
   ENUM_ACCOUNT_STOPOUT_MODE stopOutMode=(ENUM_ACCOUNT_STOPOUT_MODE)Accoun

//--- Inform about the possibility to perform a trade operation
   if(thisAccountTradeAllowed)
      Print("Trade for this account is permitted");
   else
      Print("Trade for this account is prohibited!");

//--- Find out if it is possible to trade on this account by Expert Adviso
   if(EATradeAllowed)
      Print("Trade by Expert Advisors is permitted for this account");
   else
      Print("Trade by Expert Advisors is prohibited for this account!");

//--- Find out the account type
   switch(tradeMode)
     {
      case(ACCOUNT_TRADE_MODE_DEMO):
         Print("This is a demo account");
         break;
      case(ACCOUNT_TRADE_MODE_CONTEST):
         Print("This is a competition account");
         break;
      default:Print("This is a real account!");
     }

//--- Find out the StopOut level setting mode
   switch(stopOutMode)
     {
      case(ACCOUNT_STOPOUT_MODE_PERCENT):
         Print("The StopOut level is specified percentage");
         break;
      default:Print("The StopOut level is specified in monetary terms");
     }
  }
```

See also

[Account Information]

AccountInfoString

Returns the value of the corresponding account property.

```
string  AccountInfoString(    int  property_id       // Identifier of the p
   );
```

Parameters

property_id

[in] Identifier of the property. The value can be one of the values of ENUM_ACCOUNT_INFO_STRING.

Return Value

Value of string type.

Example:

```
void OnStart()
  {
//--- Show all the information available from the function AccountInfoStri
   Print("The name of the broker = ",AccountInfoString(ACCOUNT_COMPANY));
   Print("Deposit currency = ",AccountInfoString(ACCOUNT_CURRENCY));
   Print("Client name = ",AccountInfoString(ACCOUNT_NAME));
   Print("The name of the trade server = ",AccountInfoString(ACCOUNT_SERVE
  }
```

See also

Account Information

AccountBalance

Returns balance value of the current account.

```
double  AccountBalance();
```

Returned value

Balance value of the current account (the amount of money on the account).

Example:

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

AccountCredit

Returns credit value of the current account.

```
double  AccountCredit();
```

Returned value

Credit value of the current account.

Example:

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

AccountCompany

Returns the brokerage company name where the current account was registered.

```
string  AccountCompany();
```

Returned value

The brokerage company name where the current account was registered.

Example:

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

AccountCurrency

Returns currency name of the current account.

```
string  AccountCurrency();
```

Returned value

Currency name of the current account.

Example:

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

AccountEquity

Returns equity value of the current account.

```
double  AccountEquity();
```

Returned value

Equity value of the current account. Equity calculation depends on trading server settings.

Example:

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

AccountFreeMargin

Returns free margin value of the current account.

```
double  AccountFreeMargin();
```

Returned value

Free margin value of the current account.

Example:

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

AccountFreeMarginCheck

Returns free margin that remains after the specified order has been opened at the current price on the current account.

```
double  AccountFreeMarginCheck(    string  symbol,      // symbol
   int      cmd,          // trade operation
   double   volume        // volume
   );
```

Parameters

symbol

[in] Symbol for trading operation.

cmd

[in] Operation type. It can be either OP_BUY or OP_SELL.

volume

[in] Number of lots.

Returned value

Free margin that remains after the specified order has been opened at the current price on the current account. If the free margin is insufficient, an error 134 (ERR_NOT_ENOUGH_MONEY) will be generated.

Example:

```
   if(AccountFreeMarginCheck(Symbol(),OP_BUY,Lots)<=0 || GetLastError()==13
```

AccountFreeMarginMode

Returns the calculation mode of free margin allowed to open orders on the current account.

```
double  AccountFreeMarginMode();
```

Returned value

Calculation mode of free margin allowed to opened orders on the current account.

The calculation mode can take the following values: 0 - floating profit/loss is not used for calculation;
1 - both floating profit and loss on opened orders on the current account are used for free margin calculation;
2 - only profit value is used for calculation, the current loss on opened orders is not considered;
3 - only loss value is used for calculation, the current loss on opened orders is not considered.

Example:

```
  if(AccountFreeMarginMode()==0)
     Print("Floating Profit/Loss do not use.");
```

AccountLeverage

Returns leverage of the current account.

```
int  AccountLeverage();
```

Returned value

Leverage of the current account.

Example:

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

AccountMargin

Returns margin value of the current account.

```
double  AccountMargin();
```

Returned value

Margin value of the current account.

Example:

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

AccountName

Returns the current account name.

```
string  AccountName();
```

Returned value

Name of the current account.

Example:

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

AccountNumber

Returns the current account number.

```
int  AccountNumber();
```

Returned value

The current account number.

Example:

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

AccountProfit

Returns profit value of the current account.

```
double  AccountProfit();
```

Returned value

Profit value of the current account.

Example:

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

AccountServer

Returns the connected server name.

```
string  AccountServer();
```

Returned value

Connected server name.

Example:

```
   Print("Server name is ", AccountServer());
```

AccountStopoutLevel

Returns the value of the Stop Out level.

```
int  AccountStopoutLevel();
```

Returned value

The value of the Stop Out level.

Example:

```
   Print("StopOut level = ", AccountStopoutLevel());
```

AccountStopoutMode

Returns the calculation mode for the Stop Out level.

```
int  AccountStopoutMode();
```

Returned value

Returns the calculation mode for the Stop Out level.

Calculation mode can take the following values: 0 - calculation of percentage ratio between margin and equity;
1 - comparison of the free margin level to the absolute value.

Example:

```
  int level=AccountStopoutLevel();
  if(AccountStopoutMode()==0)
     Print("StopOut level = ", level, "%");
  else
     Print("StopOut level = ", level, " ", AccountCurrency());
```

State Checking

Functions that return parameters of the current state of the client terminal

Function	Action
GetLastError	Returns the last error
IsStopped	Returns true, if an mql4 program has been commanded to stop its operation
UninitializeReason	Returns the code of the reason for deinitialization
MQLInfoInteger	Returns an integer value of a corresponding property of a running mql4 program
MQLInfoString	Returns a string value of a corresponding property of a running mql4 program
MQLSetInteger	Sets the value of the MQL_CODEPAGE property in an MQL4 program environment
TerminalInfoInteger	Returns an integer value of a corresponding property of a running mql4 program
TerminalInfoDouble	Returns a double value of a corresponding property of a running mql4 program
TerminalInfoString	Returns a string value of a corresponding property of a running mql4 program
Symbol	Returns the name of a symbol of the current chart
Period	Returns the current chart timeframe
Digits	Returns the number of decimal digits determining the accuracy of the price value of the current chart symbol
Point	Returns the point size of the current symbol in the quote currency
IsConnected	Checks connection between client terminal and server
IsDemo	Checks if the Expert Advisor runs on a demo account
IsDllsAllowed	Checks if the DLL function call is allowed for the Expert Advisor
IsExpertEnabled	Checks if Expert Advisors are enabled for running
IsLibrariesAllowed	Checks if the Expert Advisor can call library function
IsOptimization	Checks if Expert Advisor runs in the Strategy Tester optimization mode

IsTesting	Checks if the Expert Advisor runs in the testing mode
IsTradeAllowed	Checks if the Expert Advisor is allowed to trade and trading context is not busy
IsTradeContextBusy	Returns the information about trade context
IsVisualMode	Checks if the Expert Advisor is tested in visual mode
TerminalCompany	Returns the name of company owning the client terminal
TerminalName	Returns client terminal name
TerminalPath	Returns the directory, from which the client terminal was launched

GetLastError

Returns the contents of the

system variable _LastError.

```
int  GetLastError();
```

Returned Value

Returns the value of the last error that occurred during the execution of an mql4 program.

Note

After the function call, the contents of _LastError are reset.

For convenience, trade errors are additionally listed in the Trade Server Return Codes section.

IsStopped

Checks the forced shutdown of an mql4 program.

```
bool  IsStopped();
```

Returned Value

Returns true, if the _StopFlag system variable contains a value other than 0. A nonzero value is written into _StopFlag, if a mql4 program has been commanded to complete its operation. In this case, you must immediately terminate the program, otherwise the program will be completed forcibly from the outside after 3 seconds.

UninitializeReason

Returns the code of a reason for deinitialization.

```
int  UninitializeReason();
```

Returned Value

Returns the value of _UninitReason which is formed before OnDeinit() is called. Value depends on the reasons that led to deinitialization.

MQLInfoInteger

Returns the value of a corresponding property of a running mql4 program.

```
int  MQLInfoInteger(    int  property_id      // identifier of a property
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of values of the ENUM_MQL_INFO_INTEGER enumeration.

Return Value

Value of int type.

MQLInfoString

Returns the value of a corresponding property of a running MQL4 program.

```
string  MQLInfoString(    int   property_id      // Identifier of a propert
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of the ENUM_MQL_INFO_STRING enumeration.

Return Value

Value of string type.

MQLSetInteger

Sets the value of the MQL_CODEPAGE property in an MQL4 program environment.

```
void  MQLSetInteger(    int  property_id          // identifier of a prope
   int  property_value       // value to be set
   );
```

Parameters

property_id

[in] Identifier of a property. Only MQL_CODEPAGE is supported, as other properties cannot be changed.

property_value

[in] Value of property. Can be one of the Codepage constants.

Return Value

No return value

Note

The MQLSetInteger() function is intended to change current codepage in a running MQL4 program. This can be useful when the client terminal sets the default codepage that is different from the one used when the program was compiled. For example, an MQL4 program was compiled on a computer with Spanish locale, while it is running on a machine with Chinese locale.

When locales of machines the program was compiled and is running on are different, you can get errors when printing messages or getting some values. Such errors relate to the PrintFormat, Print, Comment, Alert, MessageBox, SendFTP, SendMail, SendNotification, iCustom and other functions that use object names, global variable names, etc. as their parameters.

To explicitly change a codepage to work with strings in a running program, you need to call MQLSetInteger() with required codepage passed as its second parameter. The function can be particularly useful for localization of messages displayed to user.

See also

Use of a Codepage

TerminalInfoInteger

Returns the value of a corresponding property of the mql4 program environment.

```
int  TerminalInfoInteger(    int  property_id        // identifier of a prop
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of the values of the ENUM_TERMINAL_INFO_INTEGER enumeration.

Return Value

Value of int type.

TerminalInfoDouble

Returns the value of a corresponding property of the mql4 program environment.

```
double  TerminalInfoDouble(    int  property_id      // identifier of a pr
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of the values of the ENUM_TERMINAL_INFO_DOUBLE enumeration.

Return Value

Value of double type.

TerminalInfoString

Returns the value of a corresponding property of the mql4 program environment. The property must be of string type.

```
string  TerminalInfoString(    int  property_id      // identifier of a pro
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one of the values of the ENUM_TERMINAL_INFO_STRING enumeration.

Return Value

Value of string type.

Symbol

Returns the name of a symbol of the current chart.

```
string  Symbol();
```

Returned Value

Value of the _Symbol system variable, which stores the name of the current chart symbol.

Period

Returns the current chart timeframe.

```
int  Period();
```

Returned Value

The contents of the _Period variable that contains the value of the current chart timeframe.

See also

PeriodSeconds, Chart timeframes, Date and Time, Visibility of objects

Digits

Returns the number of decimal digits determining the accuracy of price of the current chart symbol.

```
int  Digits();
```

Returned Value

The value of the _Digits variable which stores the number of decimal digits determining the accuracy of price of the current chart symbol.

Point

Returns the point size of the current symbol in the quote currency.

```
double  Point();
```

Returned Value

The value of the _Point variable which stores the point size of the current symbol in the quote currency.

IsConnected

Checks connection between client terminal and server.

```
bool  IsConnected();
```

Returned value

It returns true if connection to the server was successfully established, otherwise, it returns false.

Example:

```
  if(!IsConnected())     {
     Print("No connection!");
     return(0);
    }
  // Expert body that needs the connection opened
  // ...
```

IsDemo

Checks if the Expert Advisor runs on a demo account.

```
bool  IsDemo();
```

Returned value

Returns true if the Expert Advisor runs on a demo account, otherwise returns false.

Example:

```
   if(IsDemo()) Print("I work at a demo account");   else Print("I work at
```

IsDllsAllowed

Checks if the DLL function call is allowed for the Expert Advisor.

```
bool  IsDllsAllowed();
```

Returned value

Returns true if the DLL function call is allowed for the Expert Advisor, otherwise returns false.

Example:

```
  #import "user32.dll"     int     MessageBoxA(int hWnd, string szText, s
  ...
  ...
  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);
```

See also

IsLibrariesAllowed(), IsTradeAllowed()

IsExpertEnabled

Checks if Expert Advisors are enabled for running.

```
bool  IsExpertEnabled();
```

Returned value

Returns true if Expert Advisors are enabled for running, otherwise returns false.

Example:

```
  while(!IsStopped())     {
     ...
     if(!IsExpertEnabled()) break;
    }
```

IsLibrariesAllowed

Checks if the Expert Advisor can call library function.

```
bool  IsLibrariesAllowed();
```

Returned value

Returns true if the Expert Advisor can call library function, otherwise returns false.

Example:

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

See also

[IsDllSAllowed()](), [IsTradeAllowed()]()

IsOptimization

Checks if Expert Advisor runs in the Strategy Tester optimization mode.

```
bool  IsOptimization();
```

Returned value

Returns true if Expert Advisor runs in the Strategy Tester optimization mode, otherwise returns false.

Example:

```
   if(IsOptimization()) return(0);
```

IsTesting

Checks if the Expert Advisor runs in the testing mode.

```
bool  IsTesting();
```

Returned value

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

Example:

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

IsTradeAllowed

Checks if the Expert Advisor is allowed to trade and trading context is not busy.

```
bool  IsTradeAllowed();
```

The second form of the function checks trade status for the specified symbol in the specified time.

```
bool  IsTradeAllowed(   const string symbol       // symbol
  datetime     tested_time  // time
   );
```

Parameters

symbol

[in] Symbol.

tested_time

[in] Time to check status.

Returned value

Returns true if the Expert Advisor is allowed to trade and trading context is not busy, otherwise returns false.

Note:

OrderSend(), OrderClose(), OrderCloseBy(), OrderModify(), OrderDelete() trading functions changing the state of a trading account can be called only if trading by Expert Advisors is allowed (the "Allow live trading" checkbox is enabled in the Expert Advisor or script properties).

Example:

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

See also

IsDllsAllowed(), IsLibrariesAllowed(), IsTradeContextBusy()

IsTradeContextBusy

Returns the information about trade context.

```
bool  IsTradeContextBusy();
```

Returned value

Returns true if a thread for trading is occupied by another Expert Advisor, otherwise returns false.

Example:

```
   if(IsTradeContextBusy()) Print("Trade context is busy. Please wait");
```

IsVisualMode

Checks if the Expert Advisor is tested in visual mode.

```
bool  IsVisualMode();
```

Returned value

Returns true if the Expert Advisor is tested with checked "Visual Mode" button, otherwise returns false.

Example:

```
   if(IsVisualMode()) Comment("Visual mode turned on");
```

TerminalCompany

Returns the name of company owning the client terminal.

```
string  TerminalCompany();
```

Returned value

The name of company owning the client terminal.

Example:

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

See also

TerminalName(), TerminalPath()

TerminalName

Returns client terminal name.

```
string  TerminalName();
```

Returned value

Client terminal name.

Example:

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

See also

TerminalCompany(), TerminalPath()

TerminalPath

Returns the directory, from which the client terminal was launched.

```
string  TerminalPath();
```

Returned value

The directory, from which the client terminal was launched.

Example:

```
  Print("Working directory is ",TerminalPath());
```

See also

TerminalCompany(), TerminalName()

Getting Market Information

These are functions intended for receiving information about the market state.

Function	Action
MarketInfo	Returns various data about securities listed in the "Market Watch" window
SymbolsTotal	Returns the number of available (selected in Market Watch or all) symbols
SymbolName	Returns the name of a specified symbol
SymbolSelect	Selects a symbol in the Market Watch window or removes a symbol from the window
SymbolInfoDouble	Returns the double value of the symbol for the corresponding property
SymbolInfoInteger	Returns a value of an integer type (long, datetime, int or bool) of a specified symbol for the corresponding property
SymbolInfoString	Returns a value of the string type of a specified symbol for the corresponding property
SymbolInfoTick	Returns the current prices for the specified symbol in a variable of the MqlTick type
SymbolInfoSessionQuote	Allows receiving time of beginning and end of the specified quoting sessions for a specified symbol and day of week.
SymbolInfoSessionTrade	Allows receiving time of beginning and end of the specified trading sessions for a specified symbol and day of week.

MarketInfo

Returns various data about securities listed in the "Market Watch" window.

```
double  MarketInfo(    string              symbol,      // symbol
   int                 type         // information type
   );
```

Parameters

symbol

[in] Symbol name.

type

[in] Request identifier that defines the type of information to be returned. Can be any of values of request identifiers.

Returned value

Returns various data about securities listed in the "Market Watch" window. A part of information about the current security is stored in predefined variables.

Example:

```
   double vbid    = MarketInfo("EURUSD",MODE_BID);
   double vask    = MarketInfo("EURUSD",MODE_ASK);
   double vpoint  = MarketInfo("EURUSD",MODE_POINT);
   int    vdigits = (int)MarketInfo("EURUSD",MODE_DIGITS);
   int    vspread = (int)MarketInfo("EURUSD",MODE_SPREAD);
```

See also

Symbol properties

SymbolsTotal

Returns the number of available (selected in Market Watch or all) symbols.

```
int  SymbolsTotal(    bool  selected       // True - only symbols in MarketW
   );
```

Parameters

selected

[in] Request mode. Can be true or false.

Return Value

If the 'selected' parameter is true, the function returns the number of symbols selected in MarketWatch. If the value is false, it returns the total number of all symbols.

SymbolName

Returns the name of a symbol.

```
string  SymbolName(    int    pos,           // number in the list
   bool   selected      // true - only symbols in MarketWatch
   );
```

Parameters

pos

[in] Order number of a symbol.

selected

[in] Request mode. If the value is true, the symbol is taken from the list of symbols selected in MarketWatch. If the value is false, the symbol is taken from the general list.

Return Value

Value of string type with the symbol name.

SymbolSelect

Selects a symbol in the Market Watch window or removes a symbol from the window.

```
bool  SymbolSelect(    string  name,        // symbol name
   bool     select      // add or remove
   );
```

Parameters

name

[in] Symbol name.

select

[in] Switch. If the value is false, a symbol should be removed from MarketWatch, otherwise a symbol should be selected in this window. A symbol can't be removed if the symbol chart is open, or there are open orders for this symbol.

Return Value

In case of failure returns false.

Note

To get symbol data using functions for accessing timeseries and indicators, make sure that the symbol exists in the MarketWatch window. If the symbols is not available in Market watch, enable it using the SymbolSelect(symbol_name, true) function before you request the data.

The symbol can be hidden from MarketWatch after 10 minutes since the last access to the symbol history, i.e. since the call of functions like iOpen(), iHigh(), CopyTime() etc. This is due to the fact that the terminal stores symbol data for 10 minutes since the last access to them; after that unused data are automatically deleted from the terminal memory.

SymbolInfoDouble

Returns the corresponding property of a specified symbol. There are 2 variants of the function.

1. Immediately returns the property value.

```
double  SymbolInfoDouble(    string                         name,        // symbo
   ENUM_SYMBOL_INFO_DOUBLE  prop_id      // identifier of the property
   );
```

2. Returns true or false depending on whether a function is successfully performed. In case of success, the value of the property is placed into a recipient variable, passed by reference by the last parameter.

```
bool  SymbolInfoDouble(
   string                   name,        // symbol
   ENUM_SYMBOL_INFO_DOUBLE  prop_id,     // identifier of the property
   double&                  double_var   // here we accept the property val
   );
```

Parameters

name

[in] Symbol name.

prop_id

[in] Identifier of a symbol property. The value can be one of the values of the ENUM_SYMBOL_INFO_DOUBLE enumeration.

double_var

[out] Variable of double type receiving the value of the requested property.

Return Value

The value of double type. In case of execution failure, information about the error can be obtained using GetLastError() function:

- 4106 symbol is not selected in "Market Watch" (not found in the list of available ones),
- 4051 invalid identifier of a symbol property,
- 4024 internal error.

Note

It is recommended to use SymbolInfoTick() if the function is used for getting information about the last tick. It may well be that not a single quote has appeared yet since the terminal is connected to a trading account. In such a case, the requested value will be indefinite.

In most cases, it is enough to use SymbolInfoTick() function allowing a user to receive the values of Ask, Bid, Last, Volume and the time of the last tick's arrival during a single call.

Example:

```
void OnTick()
  {
//--- obtain spread from the symbol properties
   bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
   string comm=StringFormat("Spread %s = %I64d points\r\n",
                            spreadfloat?"floating":"fixed",
                            SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- now let's calculate the spread by ourselves
   double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   double spread=ask-bid;
   int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBO
   comm=comm+"Calculated spread = "+(string)spread_points+" points";
   Comment(comm);
  }
```

SymbolInfoInteger

Returns the corresponding property of a specified symbol. There are 2 variants of the function.

1. Immediately returns the property value.

```
long  SymbolInfoInteger(    string                    name,      // symbol
   ENUM_SYMBOL_INFO_INTEGER  prop_id    // identifier of a property

   );
```

2. Returns true or false depending on whether a function is successfully performed. In case of success, the value of the property is placed into a recipient variable, passed by reference by the last parameter.

```
bool  SymbolInfoInteger(
   string                    name,      // symbol
   ENUM_SYMBOL_INFO_INTEGER  prop_id,   // identifier of a property
   long&                     long_var   // here we accept the property val
   );
```

Parameters

name

[in] Symbol name.

prop_id

[in] Identifier of a symbol property. The value can be one of the values of the ENUM_SYMBOL_INFO_INTEGER enumeration.

long_var

[out] Variable of the long type receiving the value of the requested property.

Return Value

The value of long type. In case of execution failure, information about the error can be obtained using GetLastError() function:

- 4106 symbol is not selected in "Market Watch" (not found in the list of available ones),
- 4051 invalid identifier of a symbol property,
- 4024 internal error.

Note

It is recommended to use SymbolInfoTick() if the function is used for getting information about the last tick. It may well be that not a single quote has appeared yet since the terminal is connected to a trading account. In such a case, the requested value will be indefinite.

In most cases, it is enough to use SymbolInfoTick() function allowing a user to receive the values of Ask, Bid, Last, Volume and the time of the last tick's arrival during a single call.

Example:

```
void OnTick()
  {
//--- obtain spread from the symbol properties
   bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
   string comm=StringFormat("Spread %s = %I64d points\r\n",
                            spreadfloat?"floating":"fixed",
                            SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- now let's calculate the spread by ourselves
   double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   double spread=ask-bid;
   int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBO
   comm=comm+"Calculated spread = "+(string)spread_points+" points";
   Comment(comm);
  }
```

SymbolInfoString

Returns the corresponding property of a specified symbol. There are 2 variants of the function.

1. Immediately returns the property value.

```
string  SymbolInfoString(    string                   name,       // Symb
   ENUM_SYMBOL_INFO_STRING  prop_id      // Property identifier
   );
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a placeholder variable passed by reference in the last parameter.

```
bool  SymbolInfoString(
   string                   name,        // Symbol
   ENUM_SYMBOL_INFO_STRING  prop_id,     // Property identifier
   string&                  string_var   // Here we accept the property va
   );
```

Parameters

name

[in] Symbol name.

prop_id

[in] Identifier of a symbol property. The value can be one of the values of the ENUM_SYMBOL_INFO_STRING enumeration.

string_var

[out] Variable of the string type receiving the value of the requested property.

Return Value

The value of string type. In case of execution failure, information about the error can be obtained using GetLastError() function:

- 4106 symbol is not selected in "Market Watch" (not found in the list of available ones),
- 4051 invalid identifier of a symbol property,
- 4024 internal error.

Note

It is recommended to use SymbolInfoTick() if the function is used for getting information about the last tick. It may well be that not a single quote has appeared yet since the terminal is connected to a trading account. In such a case, the requested value will be indefinite.

In most cases, it is enough to use SymbolInfoTick() function allowing a user to receive the values of Ask, Bid, Last, Volume and the time of the last tick's arrival during a single call.

SymbolInfoTick

The function returns current prices of a specified symbol in a variable of the MqlTick type.

```
bool  SymbolInfoTick(    string    symbol,      // symbol name
   MqlTick&  tick          // reference to a structure
   );
```

Parameters

symbol

[in] Symbol name.

tick

[out] Link to the structure of the MqlTick type, to which the current prices and time of the last price update will be placed.

Return Value

The function returns true if successful, otherwise returns false.

SymbolInfoSessionQuote

Allows receiving time of beginning and end of the specified quoting sessions for a specified symbol and day of week.

```
bool  SymbolInfoSessionQuote(   string              name,             //
   ENUM_DAY_OF_WEEK  day_of_week,          // day of the week
   uint              session_index,        // session index
   datetime&         from,                 // time of the session beginning
   datetime&         to                    // time of the session end
   );
```

Parameters

name

[in] Symbol name.

ENUM_DAY_OF_WEEK

[in] Day of the week, value of enumeration ENUM_DAY_OF_WEEK.

uint

[in] Ordinal number of a session, whose beginning and end time we want to receive. Indexing of sessions starts with 0.

from

[out] Session beginning time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

to

[out] Session end time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

Return Value

If data for the specified session, symbol and day of the week are received, returns true, otherwise returns false.

See also

Symbol Properties, TimeToStruct, Data Structures

SymbolInfoSessionTrade

Allows receiving time of beginning and end of the specified trading sessions for a specified symbol and day of week.

```
bool  SymbolInfoSessionTrade(   string              name,             //
   ENUM_DAY_OF_WEEK  day_of_week,          // day of the week
   uint              session_index,        // session index
   datetime&         from,                 // session beginning time
   datetime&         to                    // session end time
   );
```

Parameters

name

[in] Symbol name.

ENUM_DAY_OF_WEEK

[in] Day of the week, value of enumeration ENUM_DAY_OF_WEEK.

uint

[in] Ordinal number of a session, whose beginning and end time we want to receive. Indexing of sessions starts with 0.

from

[out] Session beginning time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

to

[out] Session end time in seconds from 00 hours 00 minutes, in the returned value date should be ignored.

Return value

If data for the specified session, symbol and day of the week are received, returns true, otherwise returns false.

See also

Symbol Properties, TimeToStruct, Data Structures

Access to Timeseries and Indicator Data

These are functions for working with timeseries and indicators. A timeseries differs from the usual data array by its reverse ordering - elements of timeseries are indexed from the end of an array to its begin (from the most recent data to the oldest ones). To copy the time-series values and indicator data, it's recommended to use dynamic arrays only, because copying functions are designed to allocate the necessary size of arrays that receive values.

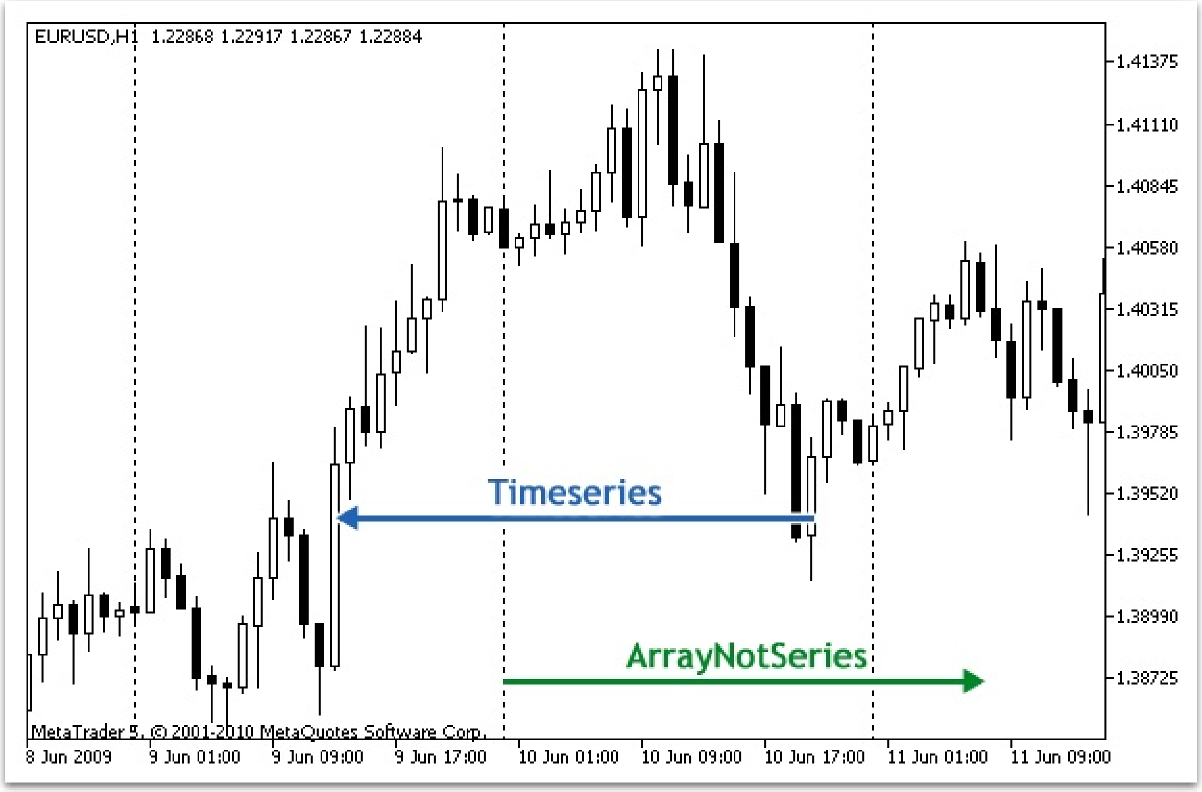
There is an **important exception** to this rule: if timeseries and indicator values need to be copied often, for example at each call of OnTick() in Expert Advisors or at each call of OnCalculate() in indicators, in this case one should better use statically distributed arrays, because **operations of memory allocation** for dynamic arrays **require additional time**, and this will have effect during testing and optimization.

When using functions accessing timeseries and indicator values, indexing direction should be taken into account. This is described in the Indexing Direction in Arrays, Buffers and Timeseries section.

Access to indicator and timeseries data is implemented irrespective of the fact whether the requested data are ready (the so called asynchronous access). This is critically important for the calculation of custom indicator, so if there are no data, functions of *Copy...()* type immediately return an error. However, when accessing form Expert Advisors and scripts, several attempts to receive data are made in a small pause, which is aimed at providing some time necessary to download required timeseries or to calculate indicator values.

If data (symbol name and/or timeframe differ from the current ones) are requested from another chart, the situation is possible that the corresponding chart was not opened in the client terminal and the necessary data must be requested from the server. In this case, error ERR_HISTORY_WILL_UPDATED (4066 - the requested history data are under updating) will be placed in the last_error variable, and one will has to re-request (see example of ArrayCopySeries()).

The Organizing Data Access section describes details of receiving, storing and requesting price data in the MetaTrader 4 client terminal.



It is historically accepted that an access to the price data in an array is performed from the end of the data. Physically, the new data are always written at the array end, but the index of the array is always equal to zero. The 0 index in the timeseries array denotes data of the current bar, i.e. the bar that corresponds to the unfinished time interval in this timeframe.

A timeframe is the time period, during which a single price bar is formed. There are several predefined standard timeframes.

Function	Action
SeriesInfoInteger	Returns information about the state of historical data
RefreshRates	Refreshing of data in pre-defined variables and series arrays
CopyRates	Gets history data of the Rates structure for a specified symbol and period into an array
CopyTime	Gets history data on bar opening time for a specified symbol and period into an array
CopyOpen	Gets history data on bar opening price for a specified symbol and period into an array
CopyHigh	Gets history data on maximal bar price for a specified symbol and period into an array
CopyLow	Gets history data on minimal bar price for a specified symbol and period into an array
CopyClose	Gets history data on bar closing price for a specified symbol and period into an array

CopyTickVolume	Gets history data on tick volumes for a specified symbol and period into an array
Bars	Returns the number of bars count in the history for a specified symbol and period
iBars	Returns the number of bars on the specified chart
iBarShift	Returns the index of the bar which covers the specified time
iClose	Returns Close price value for the bar of specified symbol with timeframe and shift
iHigh	Returns High price value for the bar of specified symbol with timeframe and shift
iHighest	Returns the shift of the maximum value over a specific number of bars
iLow	Returns Low price value for the bar of indicated symbol with timeframe and shift
iLowest	Returns the shift of the lowest value over a specific number of bars
iOpen	Returns Open price value for the bar of specified symbol with timeframe and shift
iTime	Returns time value for the bar of specified symbol with timeframe and shift
iVolume	Returns Tick Volume value for the bar of specified symbol with timeframe and shift

Despite the fact that by using the ArraySetAsSeries() function it is possible to set up in arrays access to elements like that in timeseries, it should be remembered that the array elements are physically stored in one and the same order - only indexing direction changes. To demonstrate this fact let's perform an example:

```
   datetime TimeAsSeries[]; //--- set access to the array like to a timese
   ArraySetAsSeries(TimeAsSeries,true);
   ResetLastError();
   int copied=CopyTime(NULL,0,0,10,TimeAsSeries);
   if(copied<=0)
     {
      Print("The copy operation of the open time values for last 10 bars h
      return;
     }
   Print("TimeCurrent =",TimeCurrent());
   Print("ArraySize(Time) =",ArraySize(TimeAsSeries));
   int size=ArraySize(TimeAsSeries);
   for(int i=0;i<size;i++)
     {
      Print("TimeAsSeries["+i+"] =",TimeAsSeries[i]);
     }

   datetime ArrayNotSeries[];
   ArraySetAsSeries(ArrayNotSeries,false);
   ResetLastError();
   copied=CopyTime(NULL,0,0,10,ArrayNotSeries);
   if(copied<=0)
     {
      Print("The copy operation of the open time values for last 10 bars h
      return;
     }
   size=ArraySize(ArrayNotSeries);
   for(int i=size-1;i>=0;i--)
     {
      Print("ArrayNotSeries["+i+"] =",ArrayNotSeries[i]);
     }
```

As a result we will get the output like this:

```
TimeCurrent = 2009.06.11 14:16:23
ArraySize(Time) = 10
TimeAsSeries[0] = 2009.06.11 14:00:00
TimeAsSeries[1] = 2009.06.11 13:00:00
TimeAsSeries[2] = 2009.06.11 12:00:00
TimeAsSeries[3] = 2009.06.11 11:00:00
TimeAsSeries[4] = 2009.06.11 10:00:00
TimeAsSeries[5] = 2009.06.11 09:00:00
TimeAsSeries[6] = 2009.06.11 08:00:00
TimeAsSeries[7] = 2009.06.11 07:00:00
TimeAsSeries[8] = 2009.06.11 06:00:00
TimeAsSeries[9] = 2009.06.11 05:00:00

ArrayNotSeries[9] = 2009.06.11 14:00:00
ArrayNotSeries[8] = 2009.06.11 13:00:00
ArrayNotSeries[7] = 2009.06.11 12:00:00
ArrayNotSeries[6] = 2009.06.11 11:00:00
ArrayNotSeries[5] = 2009.06.11 10:00:00
ArrayNotSeries[4] = 2009.06.11 09:00:00
ArrayNotSeries[3] = 2009.06.11 08:00:00
ArrayNotSeries[2] = 2009.06.11 07:00:00
ArrayNotSeries[1] = 2009.06.11 06:00:00
ArrayNotSeries[0] = 2009.06.11 05:00:00
```

As we see from the output, as the index of TimeAsSeries array increases, the time value of the index decreases, i.e. we move from the present to the past. For the common array ArrayNotSeries the result is different - as index grows, we move from past to present.

See Also

ArrayIsDynamic(), ArrayGetAsSeries(), ArraySetAsSeries(), ArrayIsSeries()

Indexing Direction in Arrays, Buffers and Timeseries

The default indexing of all arrays and indicator buffers is left to right. The index of the first element is always equal to zero. Thus, the very first element of an array or indicator buffer with index 0 is by default on the extreme left position, while the last element is on the extreme right position.

An indicator buffer is a dynamic array of type double, whose size is managed by the client terminals, so that it always corresponds to the number of bars the indicator is calculated on. A usual dynamic array of type double is assigned as an indicator buffer using the SetIndexBuffer() function. Indicator buffers do not require setting of their size using function ArrayResize() - this will be done by the executing system of the terminal.

Timeseries are arrays with reverse indexing, i.e. the first element of a timeseries is in the extreme right position, and the last element is in the extreme left position. Timeseries being used for storing history price data and contain the time information, we can say that the newest data are placed in the extreme right position of the timeseries, while the oldest data are in the extreme left position.

So the timeseries element with index 0 contains the information about the latest quote of a symbol. If a timeseries contains data on a daily timeframe, data of the current yet uncompleted day are located on the zero position, and the position with index 1 contains yesterday data.

Changing the Indexing Direction

Function ArraySetAsSeries() allows changing the method of accessing elements of a dynamic array; the physical order of data storing in the computer memory is not changed at that. This function simply changes the method of addressing array elements, so when copying one array to another using function ArrayCopy(), the contents of the recipient array will not depend on the indexing direction in the source array.

Direction of indexing cannot be changed for statically distributed arrays. Even if an array was passed as a parameter to a function, attempts to change the indexing direction inside this function will bring no effect.

For indicator buffers, like for usual arrays, indexing direction can also be set as backward (like in timeseries), i.e. reference to the zero position in the indicator buffer will mean reference to the last value on the corresponding indicator buffer and this will correspond to the value of the indicator on the latest bar. Still, the physical location of indicator bars will be unchanged.

Receiving Price Data in Indicators

Each custom indicator must necessarily contain the OnCalculate() function, to which price data required for calculating values in indicator buffers are passed. Indexing direction in these passed arrays can be found out using function ArrayGetAsSeries().

Arrays passed to the function reflect price data, i.e. these arrays have the sign of a timeseries and function ArrayIsSeries() will return true when checking these arrays. However, in any case indexing direction should be checked only by function ArrayGetAsSeries().

In order not to be dependent on default values, ArraySetAsSeries() should be unconditionally called for the arrays you are going to work with, and set the required direction.

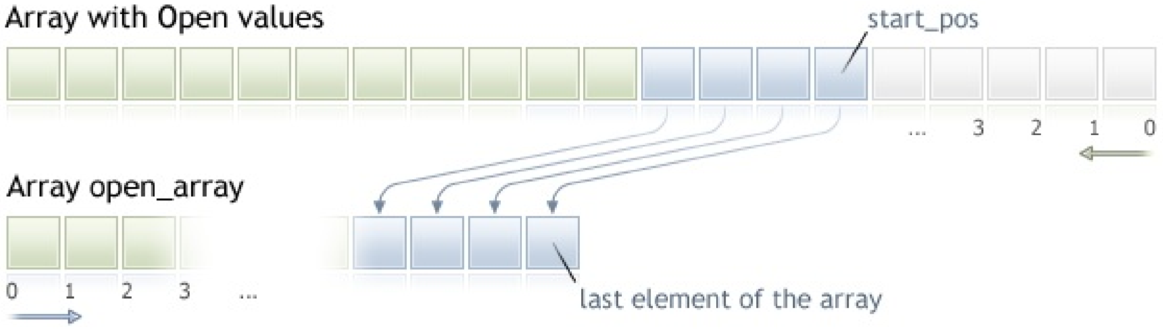
Receiving Price Data and Indicator Values

Default indexing direction of all arrays in Expert Advisors, indicators and scripts is left-to-right. If necessary, in any mql4 program you can request timeseries values on any symbol and timeframe, as well as values of indicators calculated on any symbol and timeframe.

Use functions Copy...() for these purposes:

- CopyRates copy price history to an array of structures MqlRates;
- CopyTime copy Time values to an array of datetime type;
- CopyOpen copy Open values to an array of double type;
- CopyHigh copy High values to an array of double type;
- CopyLow copy Low values to an array of double type;
- CopyClose copy Close values to an array of double type;
- CopyTickVolume copy tick volumes to an array of long type;

All these functions work in a similar way. Let's consider the data obtaining mechanism on the example of CopyOpen(). It is implied that the indexing direction of requested data is that of timeseries, and the position with index 0 (zero) stores data of the current yet uncompleted bar. In order to get access to these data we need to copy the necessary volume of data into the recipient array, e.g. into array *buffer*.



When copying we need to specify the starting position in the source array, starting from which data will be copied to the recipient array. In case of success, the specified number of elements will be copied to the recipient array from the source array (from the indicator buffer in this case). Irrespective of the indexing value set in the recipient array, copying is always performed as is shown in the above figure.

See also

Organizing Data Access

Organizing Data Access

In this section questions connected with obtaining, storing and requesting price data (timeseries) are considered.

Receiving Data from a Trade Server

Before price data become available in the MetaTrader 4 terminal, they must be received and processed. To receive data, connection to the MetaTrader 4 trade server must be established. Data are received in the form of packed blocks of minute bars from the server upon the request of a terminal.

The mechanism of server reference for data doesn't depend on how the request has been initiated - by a user when navigating in a chart or in a program way in the MQL4 language.

Storing Intermediate Data

Data received from a server are automatically unpacked and saved in the HCC intermediate format. Data on each symbol are written into a separate folder: *terminal_directory*\bases*server_name*\history*symbol_name*. For example, data on EURUSD received from the MetaQuotes-Demo server will be stored in *terminal_directory*\bases\MetaQuotes-Demo\history\EURUSD\.

Data are written into files with .hcc extension. Each file stores data of minute bars for one year. For example, the file named 2009.hcc in the EURUSD folder contains minute bars of EURUSD for year 2009. These files are used for preparing price data for all timeframes and are not intended for direct access.

Obtaining Data on a Necessary Timeframe out of Intermediate Data

Intermediate HCC files are used as the data source for building price data for requested timeframes in the HC format. Data of HC format are timeseries that are maximally prepared for a quick access. They are created upon a request of a chart or a MQL4 program. The volume of data should not exceed the value of the "Max bars in charts" parameter. Data are stored for further using in files with hc extension.

To save resources, data on a timeframe are stored and saved in RAM only if necessary. If not called for a long time, they are released from RAM and saved into a file. For each timeframe, data are prepared regardless of whether there are ready data for other timeframes or not. Rules of forming and accessing data are the same for all timeframes. I.e., despite the fact that the unit data stored in HCC is one minute (M1), the availability of HCC data doesn't mean the availability of data on M1 timeframe as HC in the same volume.

Receipt of new data from a server calls automatic update of used price data in HC format of all timeframes. It also leads to the recalculation of all indicators that implicitly use them as input data for calculations.

Parameter "Max bars in chart"

The "Max bars in charts" parameter restricts number of bars in HC format available to charts, indicators and mql45 programs. This is valid for all available timeframes and serves, first of all, to save computer resources.

When setting a large value of this parameter, it should be remembered, that if deep history price data for small timeframes are available, memory used for storing timeseries and indicator buffers can become hundreds of megabytes and reach the RAM restriction for the client terminal program (2Gb for 32-bit applications of MS Windows).

Change of the "Max bars in charts" comes into effect after the client terminal is restarted. Change of this parameter causes neither automatic referring to a server for additional data, nor forming of additional bars of timeseries. Additional price data are requested from the server, and timeseries are updated taking into account the new limitation, in case of either chart scroll to the area with no data, or when data are requested by mql4 program.

Volume of data requested from the server corresponds to the required number of bars of this timeframe with the "Max bars in charts" parameter taken into account. The restriction set by this parameter is not strict, and in some cases the number of available bars for a timeframe can be a little more than the current parameter value.

Data Availability

Presence of data on HCC format or even in the prepared for using HC format does not always denote the absolute availability if these data to be shown in a chart or used in mql4 programs.

When accessing to price data or indicator values from a mql4 program it should be remembered that their availability in a certain moment of time or starting from a certain moment of time is not guaranteed. It is connected with the fact that with the purpose of saving resources, the full copy of data necessary for a mql4 program isn't stored in MetaTrader 4; only direct access to the terminal data base is given.

The price history for all timeframes is built from common data of HCC format, and any update of data from a server leads to the update of data for all timeframes and to the recalculation of indicators. Due to this access to data can be closed, even if these data were available a moment ago.

Synchronization of the Terminal Data and Server Data

Since a mql4 program can call data fro any symbol and timeframe, there is a possibility that data of a necessary timeseries are not formed yet in the terminal or the necessary price data aren't synchronized with the trade server. In this case it's hard to predict the latency time.

Algorithms using "do-nothing" loops are not the best solution. The only exception in this case are scripts, because they do not have any alternative algorithm choice due to not having event handling. For custom indicators such algorithms, as well as any other "do-nothing" loops are strongly not recommended, because they lead to termination of calculation of all indicators and any other handling of price data of the symbol.

For Expert Advisors and indicators, it is better to use the even model of handling. If during handling of OnTick() or OnCalculate() event, data receipt for the required timeseries failed, you should exit the event handler, relying on the access availability during the next call of the handler.

SeriesInfoInteger

Returns information about the state of historical data. There are 2 variants of function calls.

Directly returns the property value.

```
long  SeriesInfoInteger(    string                           symbol_name,     //
   ENUM_TIMEFRAMES                   timeframe,        // period
   ENUM_SERIES_INFO_INTEGER          prop_id           // property identifier
   );
```

Returns true or false depending on the success of the function run.

```
bool  SeriesInfoInteger(
   string                            symbol_name,      // symbol name
   ENUM_TIMEFRAMES                   timeframe,        // period
   ENUM_SERIES_INFO_INTEGER          prop_id,          // property ID
   long&                             long_var          // variable for getting inf
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

prop_id

[in] Identifier of the requested property, value of the ENUM_SERIES_INFO_INTEGER enumeration.

long_var

[out] Variable to which the value of the requested property is placed.

Return Value

In the first case, it returns value of the long type.

For the second case, it returns true, if the specified property is available and its value has been placed into long_var variable, otherwise it returns false. For more details about an error, call GetLastError().

Example:

```
void OnStart()
  {
//---
   Print("Total number of bars for the symbol-period at this moment = ",
         SeriesInfoInteger(Symbol(),0,SERIES_BARS_COUNT));

   Print("The first date for the symbol-period at this moment = ",
         (datetime)SeriesInfoInteger(Symbol(),0,SERIES_FIRSTDATE));

   Print("The first date in the history for the symbol-period on the serve
         (datetime)SeriesInfoInteger(Symbol(),0,SERIES_SERVER_FIRSTDATE));
  }
```

RefreshRates

Refreshing of data in pre-defined variables and series arrays.

```
bool  RefreshRates();
```

Parameters

None.

Returned value

True if the data updated, otherwise false.

Note

This function is used when Expert Advisor has been calculating for a long time and needs data refreshing. Returns true if data are refreshed, otherwise returns false. The only reason for data cannot be refreshed is that they are the current data of the client terminal.

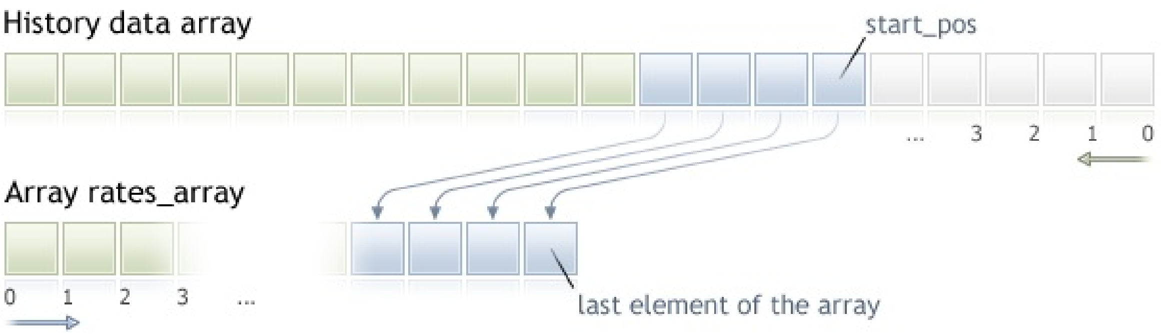
Expert Advisors and scripts operate with their own copy of history data. Data of the current symbol are copied at the first launch of the expert or script. At each subsequent launch of the expert (remember that script is executed only once and does not depend on incoming ticks), the initial copy will be updated. One or more new ticks can income while the Expert Advisor or script is operating, and data can become out of date.

Example:

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

CopyRates

Gets history data of MqlRates structure of a specified symbol-period in specified quantity into the rates_array array. The elements ordering of the copied data is from present to the past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyRates(   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   int              start_pos,         // start position
   int              count,             // data count to copy
   MqlRates         rates_array[]      // target array to copy
   );
```

Call by the start date and the number of required elements

```
int  CopyRates(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   int              count,             // data count to copy
   MqlRates         rates_array[]      // target array to copy
   );
```

Call by the start and end dates of a required time interval

```
int  CopyRates(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   datetime         stop_time,         // end date and time
   MqlRates         rates_array[]      // target array to copy
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_time

[in] Bar time for the first element to copy.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

rates_array[]

[out] Array of MqlRates type.

Return Value

Returns the number of copied elements or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

If requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

Example:

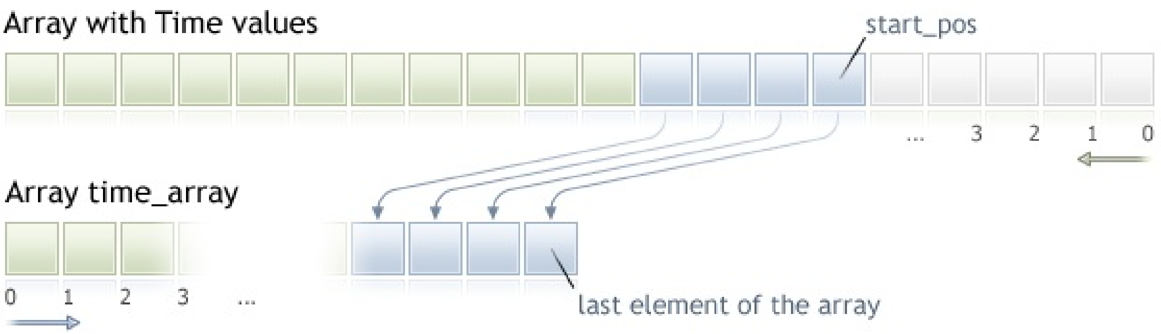
```
void OnStart()
  {
//---
   MqlRates rates[];
   ArraySetAsSeries(rates,true);
   int copied=CopyRates(Symbol(),0,0,100,rates);
   if(copied>0)
     {
      Print("Bars copied: "+copied);
      string format="open = %G, high = %G, low = %G, close = %G, volume =
      string out;
      int size=fmin(copied,10);
      for(int i=0;i<size;i++)
        {
         out=i+":"+TimeToString(rates[i].time);
         out=out+" "+StringFormat(format,
                                  rates[i].open,
                                  rates[i].high,
                                  rates[i].low,
                                  rates[i].close,
                                  rates[i].tick_volume);
         Print(out);
        }
     }
   else Print("Failed to get history data for the symbol ",Symbol());
  }
```

See also

Structures and Classes, TimeToString, StringFormat

CopyTime

The function gets to time_array history data of bar opening time for the specified symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyTime(    string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   int              start_pos,       // start position
   int              count,           // data count to copy
   datetime         time_array[]     // target array to copy open times
   );
```

Call by the start date and the number of required elements

```
int  CopyTime(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   datetime         start_time,      // start date and time
   int              count,           // data count to copy
   datetime         time_array[]     // target array to copy  open times
   );
```

Call by the start and end dates of a required time interval

```
int  CopyTime(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   datetime         start_time,      // start date and time
   datetime         stop_time,       // stop date and time
   datetime         time_array[]     // target array to copy open times
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time corresponding to the last element to copy.

time_array[]

[out] Array of datetime type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

If requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

See a detailed example of requesting history data in section Methods of Object Binding. The script available in that section shows how to get the values of indicator iFractals on the last 1000 bars and how to display the last 10 up and 10 down fractals on the chart. A similar technique can be used for all indicators that have missing data and that are usually drawn using the following styles:

· DRAW_SECTION,

· DRAW_ARROW,

· DRAW_ZIGZAG,

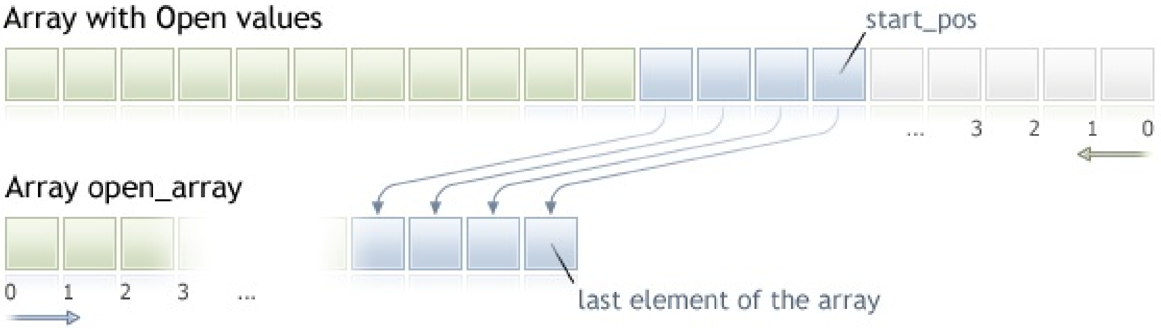
· DRAW_COLOR_SECTION,

· DRAW_COLOR_ARROW,

· DRAW_COLOR_ZIGZAG.

CopyOpen

The function gets into open_array the history data of bar open prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyOpen(    string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   int              start_pos,       // start position
   int              count,           // data count to copy
   double           open_array[]     // target array to copy open prices
   );
```

Call by the start date and the number of required elements

```
int  CopyOpen(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   int              count,            // data count to copy
   double           open_array[]      // target array for bar open prices
   );
```

Call by the start and end dates of a required time interval

```
int  CopyOpen(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   datetime         stop_time,        // stop date and time
   double           open_array[]      // target array for bar open values
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] The start time for the last element to copy.

open_array[]

[out] Array of double type.

Return Value

Returns the number of element in the array or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

If requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

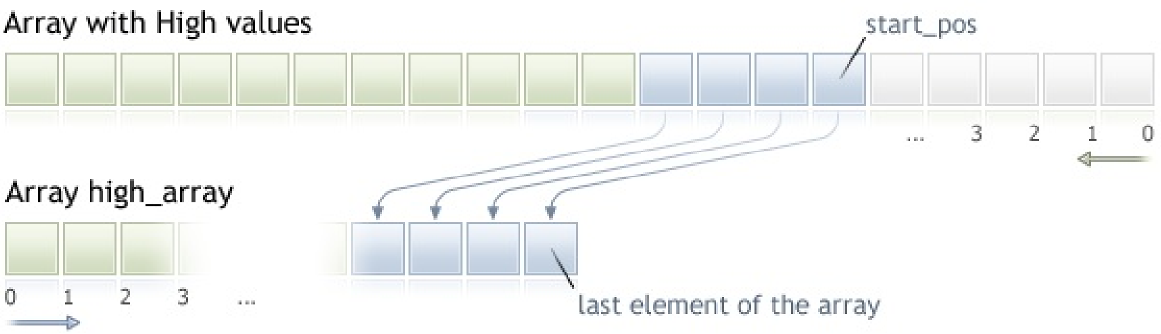
When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

CopyHigh

The function gets into high_array the history data of highest bar prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyHigh(    string             symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   int              start_pos,         // start position
   int              count,             // data count to copy
   double           high_array[]       // target array to copy
   );
```

Call by the start date and the number of required elements

```
int  CopyHigh(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   int              count,            // data count to copy
   double           high_array[]      // target array to copy
   );
```

Call by the start and end dates of a required time interval

```
int  CopyHigh(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   datetime         stop_time,        // stop date and time
   double           high_array[]      // target array to copy
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

high_array[]

[out] Array of double type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

If requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

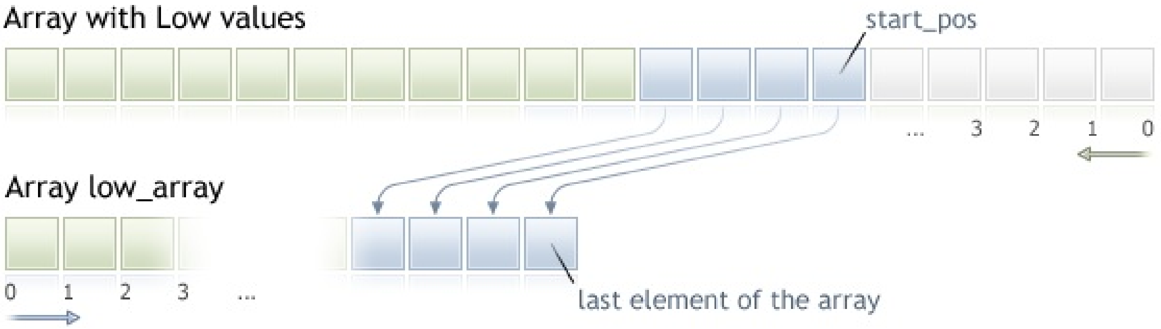
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

Example:

```
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

#property description "An example for output of the High[i] and Low[i]"
#property description "for a random chosen bars"

double High[],Low[];
//+------------------------------------------------------------------+
//| Get Low for specified bar index                                  |
//+------------------------------------------------------------------+
double iLow(string symbol,ENUM_TIMEFRAMES timeframe,int index)
  {
   double low=0;
   ArraySetAsSeries(Low,true);
   int copied=CopyLow(symbol,timeframe,0,Bars(symbol,timeframe),Low);
   if(copied>0 && index<copied) low=Low[index];
   return(low);
  }
//+------------------------------------------------------------------+
//| Get the High for specified bar index                             |
//+------------------------------------------------------------------+
double iHigh(string symbol,ENUM_TIMEFRAMES timeframe,int index)
  {
   double high=0;
   ArraySetAsSeries(High,true);
   int copied=CopyHigh(symbol,timeframe,0,Bars(symbol,timeframe),High);
   if(copied>0 && index<copied) high=High[index];
   return(high);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- on every tick we output the High and Low values for the bar with ind
//--- that is equal to the second, on which tick arrived
   datetime t=TimeCurrent();
   int sec=t%60;
   printf("High[%d] = %G  Low[%d] = %G",
          sec,iHigh(Symbol(),0,sec),
          sec,iLow(Symbol(),0,sec));
  }
```

CopyLow

The function gets into low_array the history data of minimal bar prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyLow(    string              symbol_name,     // symbol name
   ENUM_TIMEFRAMES   timeframe,         // period
   int               start_pos,         // start position
   int               count,             // data count to copy
   double            low_array[]        // target array to copy
   );
```

Call by the start date and the number of required elements

```
int  CopyLow(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   datetime         start_time,      // start date and time
   int              count,           // data count to copy
   double           low_array[]      // target array to copy
   );
```

Call by the start and end dates of a required time interval

```
int  CopyLow(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   datetime         start_time,      // start date and time
   datetime         stop_time,       // stop date and time
   double           low_array[]      // target array to copy
   );
```

Parameters

symbol_name

[in] Symbol.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] Bar time, corresponding to the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

low_array[]

[out] Array of double type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

If requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

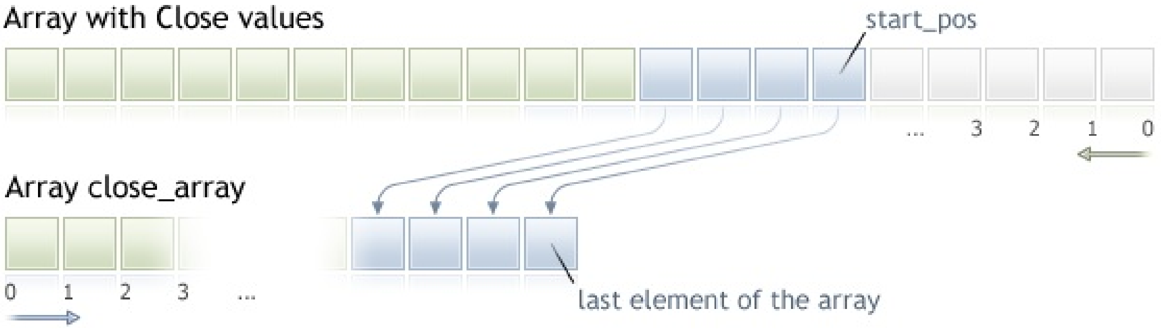
If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

See also

CopyHigh

CopyClose

The function gets into close_array the history data of bar close prices for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyClose(    string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   int              start_pos,         // start position
   int              count,             // data count to copy
   double           close_array[]      // target array to copy
   );
```

Call by the start date and the number of required elements

```
int  CopyClose(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   int              count,             // data count to copy
   double           close_array[]      // target array to copy
   );
```

Call by the start and end dates of a required time interval

```
int  CopyClose(
   string           symbol_name,       // symbol name
   ENUM_TIMEFRAMES  timeframe,         // period
   datetime         start_time,        // start date and time
   datetime         stop_time,         // stop date and time
   double           close_array[]      // target array to copy
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

close_array[]

[out] Array of double type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

If requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

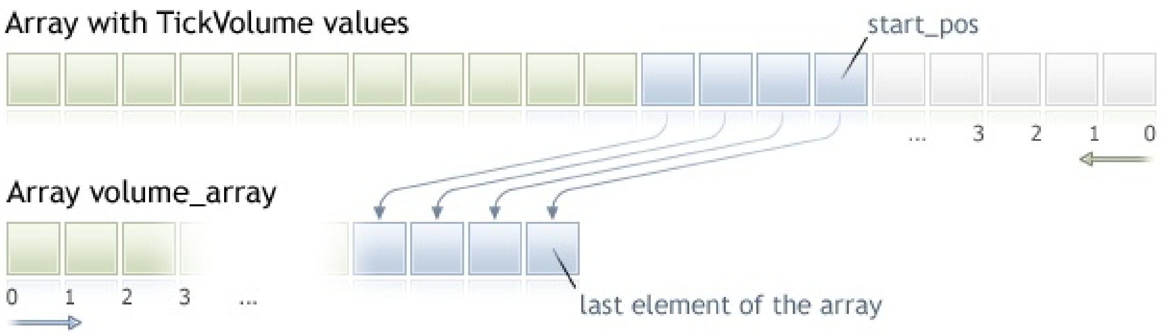
When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

CopyTickVolume

The function gets into volume_array the history data of tick volumes for the selected symbol-period pair in the specified quantity. It should be noted that elements ordering is from present to past, i.e., starting position of 0 means the current bar.



When copying the yet unknown amount of data, it is recommended to use dynamic array as a target array, because if the requested data count is less (or more) than the length of the target array, function tries to reallocate the memory so that the requested data fit entirely.

If you know the amount of data you need to copy, it should better be done to a statically allocated buffer, in order to prevent the allocation of excessive memory.

No matter what is the property of the target array - as_series=true or as_series=false. Data will be copied so that the oldest element will be located at the start of the physical memory allocated for the array. There are 3 variants of function calls.

Call by the first position and the number of required elements

```
int  CopyTickVolume(    string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   int              start_pos,        // start position
   int              count,            // data count to copy
   long             volume_array[]    // target array for tick volumes
   );
```

Call by the start date and the number of required elements

```
int  CopyTickVolume(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   int              count,            // data count to copy
   long             volume_array[]    // target array for tick volumes
   );
```

Call by the start and end dates of a required time interval

```
int  CopyTickVolume(
   string           symbol_name,      // symbol name
   ENUM_TIMEFRAMES  timeframe,        // period
   datetime         start_time,       // start date and time
   datetime         stop_time,        // stop date and time
   long             volume_array[]    // target array for tick volumes
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_pos

[in] The start position for the first element to copy.

count

[in] Data count to copy.

start_time

[in] The start time for the first element to copy.

stop_time

[in] Bar time, corresponding to the last element to copy.

volume_array[]

[out] Array of long type.

Return Value

Returns the copied data count or -1 in case of an error.

Note

If the whole interval of requested data is out of the available data on the server, the function returns -1. If data outside TERMINAL_MAXBARS (maximal number of bars on the chart) is requested, the function will also return -1.

If requested timeseries are not yet built or they need to be downloaded from the server, the function will immediately return -1.

When requesting data by the start date and the number of required elements, only data whose date is less than (earlier) or equal to the date specified will be returned. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always less or equal to the specified one.

When requesting data in a specified range of dates, only data from this interval will be returned. The interval is set and counted up to seconds. It means, the open time of any bar, for which value is returned (volume, spread, value on the indicator buffer, prices Open, High, Low, Close or open time Time) is always within the requested interval.

Thus, if the current day is Saturday, at the attempt to copy data on a week timeframe specifying *start_time=Last_Tuesday* and *stop_time=Last_Friday* the function will return 0, because the open time on a week timeframe is always Sunday, but one week bar does not fall into the specified interval.

If you need to return value corresponding to the current uncompleted bar, you can use the first form of call specifying *start_pos*=0 and *count*=1.

Example:

```
#property strict
#property indicator_separate_window
#property indicator_buffers 1
//---- plot TickVolume
#property indicator_label1  "TickVolume"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  C'143,188,139'
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      bars=3000;
//--- indicator buffers
double         TickVolumeBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,TickVolumeBuffer,INDICATOR_DATA);
   IndicatorSetInteger(INDICATOR_DIGITS,0);
```

```
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   if(prev_calculated==0)
     {
      long timeseries[];
      ArraySetAsSeries(timeseries,true);
      int copied=CopyTickVolume(Symbol(),0,0,bars,timeseries);
      for(int i=rates_total-copied-1;i>copied-1;i--) TickVolumeBuffer[i]=0
      for(int i=0;i<copied;i++) TickVolumeBuffer[i]=(double)timeseries[i];
      Print("We have received the following number of TickVolume values: "
     }
   else
     {
      long timeseries[];
      int copied=CopyTickVolume(Symbol(),0,0,1,timeseries);
      TickVolumeBuffer[0]=(double)timeseries[0];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Bars

Returns the number of bars count in the history for a specified symbol and period. There are 2 variants of functions calls.

Request all of the history bars

```
int  Bars(    string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe         // period
   );
```

Request the history bars for the selected time interval

```
int  Bars(
   string           symbol_name,     // symbol name
   ENUM_TIMEFRAMES  timeframe,       // period
   datetime         start_time,      // start date and time
   datetime         stop_time        // end date and time
   );
```

Parameters

symbol_name

[in] Symbol name.

timeframe

[in] Period.

start_time

[in] Bar time corresponding to the first element.

stop_time

[in] Bar time corresponding to the last element.

Return Value

If the start_time and stop_time parameters are defined, the function returns the number of bars in the specified time interval, otherwise it returns the total number of bars.

Note

If data for the timeseries with specified parameters are not formed in the terminal by the time of the Bars() function call, or data of the timeseries are not synchronized with a trade server by the moment of the function call, the function returns a zero value.

See also

Predefined variable Bars, iBars

iBars

Returns the number of bars on the specified chart.

```
int  iBars(    string             symbol,          // symbol
   int                timeframe          // timeframe
   );
```

Parameters

symbol

[in] Symbol the data of which should be used to calculate indicator. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

Returned value

The number of bars on the specified chart.

Note

For the current chart, the information about the amount of bars is in the Bars predefined variable.

Example:

```
   Print("Bar count on the 'EURUSD,H1' is ",iBars("EURUSD",PERIOD_H1));
```

iBarShift

Search for a bar by its time. The function returns the index of the bar which covers the specified time.

```
int  iBarShift(    string              symbol,            // symbol
   int                 timeframe,         // timeframe
   datetime            time,              // time
   bool                exact=false        // mode
   );
```

Parameters

symbol

[in] Symbol name. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

time

[in] Time value for searching.

exact=false

[in] Return mode when the bar is not found (false - iBarShift returns the nearest, true - iBarShift returns -1).

Returned value

Index of the bar which covers the specified time. If there is no bar for the specified time (history "gap"), the function will return -1 or the nearest bar index (depending on *exact* parameter).

Example:

```
  datetime some_time=D'2004.03.21 12:00';
  int      shift=iBarShift("EURUSD",PERIOD_M1,some_time);
  Print("index of the bar for the time ",TimeToStr(some_time)," is ",shift
```

iClose

Returns Close price value for the bar of specified symbol with timeframe and shift.

```
double  iClose(    string          symbol,           // symbol
   int                 timeframe,         // timeframe
   int                 shift              // shift
   );
```

Parameters

symbol

[in] Symbol name. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Close price value for the bar of specified symbol with timeframe and shift. If local history is empty (not loaded), function returns 0. To check errors, one has to call the GetLastError() function.

Note

For the current chart, the information about close prices is in the Close[] predefined array.

Example:

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,0),", ",   i
                                      iHigh("USDCHF",PERIOD_H1,0),", ",   i
                                      iClose("USDCHF",PERIOD_H1,0),", ",  i
```

iHigh

Returns High price value for the bar of specified symbol with timeframe and shift.

```
double  iHigh(    string             symbol,           // symbol
   int                 timeframe,        // timeframe
   int                 shift             // shift
   );
```

Parameters

symbol

[in] Symbol name. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

High value for the bar of specified symbol with timeframe and shift. If local history is empty (not loaded), function returns 0. To check errors, one has to call the GetLastError() function.

Note

For the current chart, the information about high prices is in the High[] predefined array.

Example:

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,0),", ",  i
                                      iHigh("USDCHF",PERIOD_H1,0),", ",  i
                                      iClose("USDCHF",PERIOD_H1,0),", ", i
```

iHighest

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

```
int  iHighest(    string           symbol,          // symbol
   int                timeframe,         // timeframe
   int                type,              // timeseries
   int                count,             // cont
   int                start              // start
  );
```

Parameters

symbol

[in] Symbol the data of which should be used for search. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

type

[in] Series array identifier. It can be any of the Series array identifier enumeration values.

count=WHOLE_ARRAY

[in] Number of bars (in direction from the start bar to the back one) on which the search is carried out.

start=0

[in] Shift showing the bar, relative to the current bar, that the data should be taken from.

Returned value

The shift of the maximum value over a specific number of bars or -1 if error. To check errors, one has to call the GetLastError() function.

Example:

```
  double val;
//--- calculating the highest value on the 20 consecutive bars in the rang
//--- from the 4th to the 23rd index inclusive on the current chart
   int val_index=iHighest(NULL,0,MODE_HIGH,20,4);
   if(val_index!=-1) val=High[val_index];
   else PrintFormat("Error in call iHighest. Error code=%d",GetLastError()
```

iLow

Returns Low price value for the bar of indicated symbol with timeframe and shift.

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

Parameters

symbol

[in] Symbol name. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Low price value for the bar of specified symbol with timeframe and shift. If local history is empty (not loaded), function returns 0. To check errors, one has to call the GetLastError() function.

Note

For the current chart, the information about low prices is in the Low[] predefined array.

Example:

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,0),", ",  i
                                      iHigh("USDCHF",PERIOD_H1,0),", ",  i
                                      iClose("USDCHF",PERIOD_H1,0),", ", i
```

iLowest

Returns the shift of the lowest value over a specific number of bars depending on type.

```
int  iLowest(     string          symbol,          // symbol
   int                timeframe,       // timeframe
   int                type,            // timeseries id
   int                count,           // count
   int                start            // starting index
  );
```

Parameters

symbol

[in] Symbol name. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

type

[in] Series array identifier. It can be any of the Series array identifier enumeration values.

count=WHOLE_ARRAY

[in] Number of bars (in direction from the start bar to the back one) on which the search is carried out.

start=0

[in] Shift showing the bar, relative to the current bar, that the data should be taken from.

Returned value

The shift of the lowest value over a specific number of bars or -1 if error. To check errors, one has to call the GetLastError() function.

Example:

```
   double val;
//--- calculating the lowest value on the 10 consequtive bars in the range
//--- from the 10th to the 19th index inclusive on the current chart
   int val_index=iLowest(NULL,0,MODE_LOW,10,10);
   if(val_index!=-1) val=Low[val_index];
   else PrintFormat("Error in iLowest. Error code=%d",GetLastError());
```

iOpen

Returns Open price value for the bar of specified symbol with timeframe and shift.

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

Parameters

symbol

[in] Symbol name. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Open price value for the bar of specified symbol with timeframe and shift or 0 if error. To check errors, one has to call the GetLastError() function.

Note

For the current chart, the information about open prices is in the Open[] predefined array.

Example:

```
   Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,0),", ",  i
                                       iHigh("USDCHF",PERIOD_H1,0),", ",  i
                                       iClose("USDCHF",PERIOD_H1,0),", ", i
```

iTime

Returns Time value for the bar of specified symbol with timeframe and shift.

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

Parameters

symbol

[in] Symbol name. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Time value for the bar of specified symbol with timeframe and shift. If local history is empty (not loaded), function returns 0. To check errors, one has to call the GetLastError() function.

Note

For the current chart, the information about open bar times is in the Time[] predefined array.

Example:

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,0),", ", i
                                      iHigh("USDCHF",PERIOD_H1,0),", ", i
                                      iClose("USDCHF",PERIOD_H1,0),", ", i
```

iVolume

Returns Tick Volume value for the bar of specified symbol with timeframe and shift.

```
long  iVolume(    string              symbol,             // symbol
   int                  timeframe,         // timeframe
   int                  shift              // shift
   );
```

Parameters

symbol

[in] Symbol name. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Tick volume value for the bar of specified symbol with timeframe and shift. If local history is empty (not loaded), function returns 0. To check errors, one has to call the GetLastError() function.

Note

For the current chart, the information about bars tick volumes is in the Volume[] predefined array.

Example:

```
  Print("Current bar for USDCHF H1: ",iTime("USDCHF",PERIOD_H1,0),", ",  i
                                      iHigh("USDCHF",PERIOD_H1,0),", ",  i
                                      iClose("USDCHF",PERIOD_H1,0),", ", i
```

Chart Operations

These are functions for working with charts. All chart operations are allowed in Expert Advisors and scripts only.

The functions defining the chart properties are actually used for sending change commands to the chart. If these functions are executed successfully, the command is included in the common queue of the chart events. The changes are implemented to the chart when handling the queue of the chart events.

Thus, do not expect an immediate visual update of the chart after calling these functions. Generally, the chart is updated automatically by the terminal following the change events - a new quote arrival, resizing the chart window, etc. Use ChartRedraw() function to forcefully update the chart.

Function	Action
ChartApplyTemplate	Applies a specific template from a specified file to the chart
ChartSaveTemplate	Saves current chart settings in a template with a specified name
ChartWindowFind	Returns the number of a subwindow where an indicator is drawn
ChartTimePriceToXY	Converts the coordinates of a chart from the time/price representation to the X and Y coordinates
ChartXYToTimePrice	Converts the X and Y coordinates on a chart to the time and price values
ChartOpen	Opens a new chart with the specified symbol and period
ChartFirst	Returns the ID of the first chart of the client terminal
ChartNext	Returns the chart ID of the chart next to the specified one
ChartClose	Closes the specified chart
ChartSymbol	Returns the symbol name of the specified chart
ChartPeriod	Returns the period value of the specified chart
ChartRedraw	Calls a forced redrawing of a specified chart
ChartSetDouble	Sets the double value for a corresponding property of the specified chart
ChartSetInteger	Sets the integer value (datetime, int, color, bool or char) for a corresponding property of the specified chart

ChartSetString	Sets the string value for a corresponding property of the specified chart
ChartGetDouble	Returns the double value property of the specified chart
ChartGetInteger	Returns the integer value property of the specified chart
ChartGetString	Returns the string value property of the specified chart
ChartNavigate	Performs shift of the specified chart by the specified number of bars relative to the specified position in the chart
ChartID	Returns the ID of the current chart
ChartIndicatorDelete	Removes an indicator with a specified name from the specified chart window
ChartIndicatorName	Returns the short name of the indicator by the number in the indicators list on the specified chart window
ChartIndicatorsTotal	Returns the number of all indicators applied to the specified chart window.
ChartWindowOnDropped	Returns the number (index) of the chart subwindow the Expert Advisor or script has been dropped to
ChartPriceOnDropped	Returns the price coordinate of the chart point the Expert Advisor or script has been dropped to
ChartTimeOnDropped	Returns the time coordinate of the chart point the Expert Advisor or script has been dropped to
ChartXOnDropped	Returns the X coordinate of the chart point the Expert Advisor or script has been dropped to
ChartYOnDropped	Returns the Y coordinate of the chart point the Expert Advisor or script has been dropped to
ChartSetSymbolPeriod	Changes the symbol value and a period of the specified chart
ChartScreenShot	Provides a screenshot of the chart of its current state in a gif format
Period	Returns timeframe of the current chart
Symbol	Returns a text string with the name of the current financial instrument
WindowBarsPerChart	Returns the amount of bars visible on the chart
WindowExpertName	Returns the name of the executed Expert Advisor, script, custom indicator, or library
WindowFind	Returns the window index containing this specified indicator
WindowFirstVisibleBar	Returns index of the first visible bar in the current chart

	window
WindowHandle	Returns the system handle of the chart window
WindowIsVisible	Returns the visibility flag of the chart subwindow
WindowOnDropped	Returns the window index where Expert Advisor, custom indicator or script was dropped
WindowPriceMax	Returns the maximal value of the vertical scale of the specified subwindow of the current chart
WindowPriceMin	Returns the minimal value of the vertical scale of the specified subwindow of the current chart
WindowPriceOnDropped	Returns the price of the chart point where Expert Advisor or script was dropped
WindowRedraw	Redraws the current chart forcedly
WindowScreenShot	Saves current chart screen shot as a GIF, PNG or BMP file depending on specified extension
WindowTimeOnDropped	Returns the time of the chart point where Expert Advisor or script was dropped
WindowsTotal	Returns total number of indicator windows on the chart
WindowXOnDropped	Returns the value at X axis in pixels for the chart window client area point at which the Expert Advisor or script was dropped
WindowYOnDropped	Returns the value at Y axis in pixels for the chart window client area point at which the Expert Advisor or script was dropped

ChartApplyTemplate

Applies a specific template from a specified file to the chart. The command is added to chart message queue and executed only after all previous commands have been processed.

```
bool  ChartApplyTemplate(    long            chart_id,      // Chart ID
   const string   filename      // Template file name
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

filename

[in] The name of the file containing the template.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get error details use the GetLastError() function.

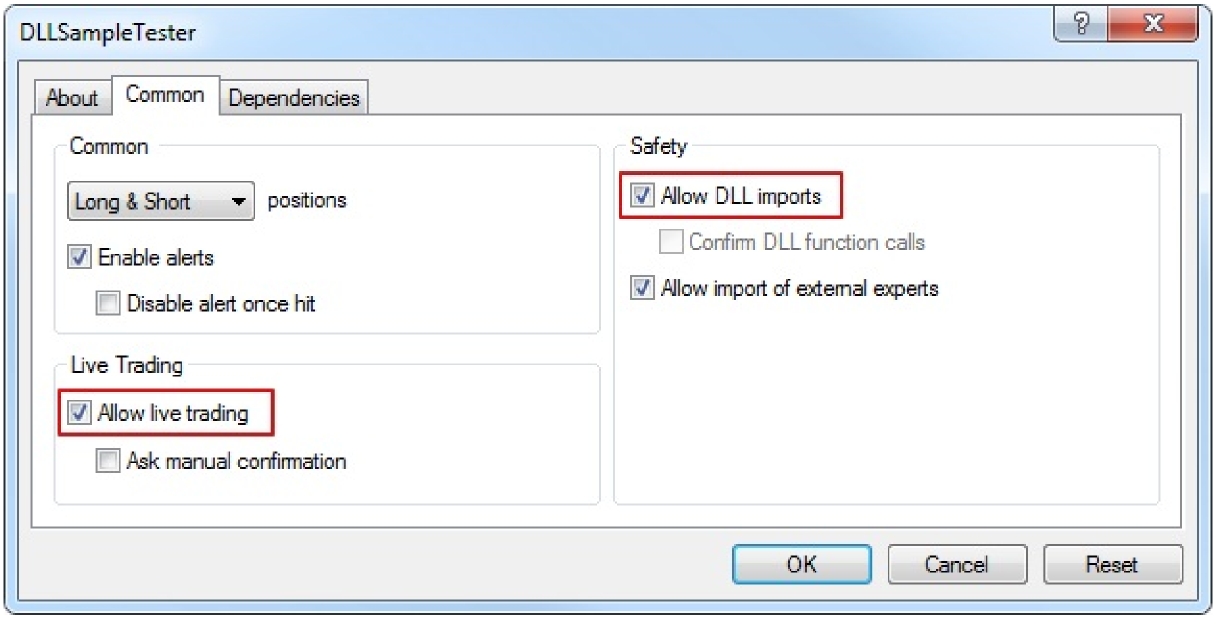
Note

The Expert Advisor will be unloaded and will not be able to continue operating in case of successful loading of a new template to the chart it is attached to.

Live Trading and DLL Imports

Users can allow or forbid the following actions for an mql4 program launched on a chart:

1.Performing trade operations (live trading),
2.DLL imports.



Common

Long & Short positions

☑ Enable alerts

□ Disable alert once hit

Live Trading

☑ Allow live trading

□ Ask manual confirmation

Safety

☑ Allow DLL imports

□ Confirm DLL function calls

☑ Allow import of external experts

The terminal allows saving the configured chart as a template with all indicators and Expert Advisors launched on it. Thus, it is possible to quickly apply template settings to all other charts. When saving the template, permissions of the launched programs (live trading and DLL imports) are also saved. When applying the template to the chart, these permissions may be limited due to security reasons:

> **Live trading and DLL imports permissions cannot be extended for the Expert Advisors launched by applying the template using ChartApplyTemplate() function.**

If the mql4 program calling ChartApplyTemplate() function has no permission to trade, the Expert Advisor launched via the template will also not be able to trade regardless of the template settings.

If the mql4 program calling ChartApplyTemplate() function has permission to trade, while there is no such permission in the template settings, the Expert Advisor launched via the template will not be able to trade.

Using Templates

The resources of the MQL4 language allow setting multiple chart properties, including colors using the ChartSetInteger() function:

- Chart background color;
- Color of the axes, scale and the OHLC line;
- Grid color;
- Color of volumes and position open levels;
- Color of the up bar, shadow and edge of a bullish candlestick;

- Color of the down bar, shadow and edge of a bearish candlestick;
- Color of the chart line and Doji candlesticks;
- Color of the bullish candlestick body;
- Color of the bearish candlestick body;
- Color of the Bid price line;
- Color of the Ask price line;
- Color of the line of the last deal price (Last);
- Color of the stop order levels (Stop Loss and Take Profit).

Besides, there can be multiple graphical objects and indicators on a chart. You may set up a chart with all the necessary indicators once and then save it as a template. Such a template can be applied to any chart.

The ChartApplyTemplate() function is intended for using a previously saved template, and it can be used in any mql4 program. The path to the file that stores the template is passed as the second parameter to ChartApplyTemplate(). The template file is searched according to the following rules:

- if the backslash "\" separator (written as "\\") is placed at the beginning of the path, the template is searched for relative to the path _terminal_data_directory\MQL4,
- if there is no backslash, the template is searched for relative to the executable EX4 file, in which ChartApplyTemplate() is called;
- if a template is not found in the first two variants, the search is performed in the folder terminal_directory\Profiles\Templates\.

Here terminal_directory is the folder from which the MetaTrader 4 Client Terminal is running, and terminal_data_directory is the folder, in which editable files are stored, its location depends on the operating system, user name and computer's security settings. Normally they are different folders, but in some cases they may coincide.

The location of folders terminal_data_directory and terminal_directory can be obtained using the TerminalInfoString() function.

```
//--- directory from which the terminal is started
   string terminal_path=TerminalInfoString(TERMINAL_PATH);
   Print("Terminal directory:",terminal_path);
//--- terminal data directory, in which the MQL4 folder with EAs and indic
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
   Print("Terminal data directory:",terminal_data_path);
```

For example:

```
//--- search for a template in terminal_data_directory\MQL4\
ChartApplyTemplate(0,"\\first_template.tpl"))

//--- search for a template in directory_of_EX4_file\, then in folder term
ChartApplyTemplate(0,"second_template.tpl"))

//--- search for a template in directory_of_EX4_file\My_templates\, then i
ChartApplyTemplate(0,"My_templates\\third_template.tpl"))
```

Templates are not resources, they cannot be included into an executable EX4 file.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- example of applying template, located in \MQL4\Files
   if(FileIsExist("my_template.tpl"))
     {
      Print("The file my_template.tpl found in \Files'");
      //--- apply template
      if(ChartApplyTemplate(0,"\\Files\\my_template.tpl"))
        {
         Print("The template 'my_template.tpl' applied successfully");
        }
      else
         Print("Failed to apply 'my_template.tpl', error code ",GetLastErr
     }
   else
     {
      Print("File 'my_template.tpl' not found in "
            +TerminalInfoString(TERMINAL_PATH)+"\\MQL4\\Files");
     }
  }
```

See also

Resources

ChartSaveTemplate

Saves current chart settings in a template with a specified name. The command is added to chart message queue and executed only after all previous commands have been processed.

```
bool  ChartSaveTemplate(    long            chart_id,     // Chart ID
   const string   filename       // Filename to save the template
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

filename

[in] The filename to save the template. The ".tpl" extension will be added to the filename automatically; there is no need to specify it. The template is saved in **terminal_directory**\Profiles\Templates\ and can be used for manual application in the terminal. If a template with the same filename already exists, the contents of this file will be overwritten.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get error details use the GetLastError() function.

Note

Using templates, you can save chart settings with all applied indicators and graphical objects, to then apply it to another chart.

Example:

```
//+------------------------------------------------------------------+
//|                                       Test_ChartSaveTemplate.mq4 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input string               symbol="GBPUSD";  // The symbol of a new chart
input ENUM_TIMEFRAMES      period=PERIOD_H3; // The timeframe of a new cha
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- First attach indicators to the chart
   int handle;
//--- Prepare the indicator for use
   if(!PrepareZigzag(NULL,0,handle)) return; // Failed, so exit
//--- Attach the indicator to the current chart, but in a separate window.
   if(!ChartIndicatorAdd(0,1,handle))
     {
      PrintFormat("Failed to attach to chart %s/%s an indicator with the h
                  _Symbol,
                  EnumToString(_Period),
                  handle,
                  GetLastError());
      //--- Terminate the program operation
      return;
     }
//--- Refresh the chart to see the indicator
   ChartRedraw();
//--- Find the last two last fractures of the zigzag
   double two_values[];
   datetime two_times[];
   if(!GetLastTwoFractures(two_values,two_times,handle))
     {
      PrintFormat("Failed to find two last fractures in the Zigzag!");
      //--- Terminate the program operation
      return;
     }
//--- Now attach a standard deviation channel
   string channel="StdDeviation Channel";
   if(!ObjectCreate(0,channel,OBJ_STDDEVCHANNEL,0,two_times[1],0))
     {
      PrintFormat("Failed to create object %s. Error code %d",
                  EnumToString(OBJ_STDDEVCHANNEL),GetLastError());
      return;
     }
   else
     {
      //--- The channel has been created, define the second point
      ObjectSetInteger(0,channel,OBJPROP_TIME,1,two_times[0]);
      //--- Set a tooltip text for the channel
      ObjectSetString(0,channel,OBJPROP_TOOLTIP,"Demo from MQL4 Help");
      //--- Refresh the chart
      ChartRedraw();
     }
```

```
//--- Save the result in a template
   ChartSaveTemplate(0,"StdDevChannelOnZigzag");
//--- Open a new chart and apply a saved template to it
   long new_chart=ChartOpen(symbol,period);
   //--- Enable tooltips for graphical objects
   ChartSetInteger(new_chart,CHART_SHOW_OBJECT_DESCR,true);
   if(new_chart!=0)
     {
      //--- Apply the saved template to a chart
      ChartApplyTemplate(new_chart,"StdDevChannelOnZigzag");
     }
   Sleep(10000);
  }
//+------------------------------------------------------------------+
//| Creates a zigzag handle and ensures readiness of its data        |
//+------------------------------------------------------------------+
bool PrepareZigzag(string sym,ENUM_TIMEFRAMES tf,int &h)
  {
   ResetLastError();
//--- The Zigzag indicator must be located in terminal_data_folder\MQL4\Ex
   h=iCustom(sym,tf,"Examples\\Zigzag");
   if(h==INVALID_HANDLE)
     {
      PrintFormat("%s: Failed to create the handle of the Zigzag indicator
                  __FUNCTION__,GetLastError());
      return false;
     }
//--- When creating an indicator handle, it requires time to calculate val
   int k=0; // The number of attempts to wait for the indicator calculatio
//--- Wait for the calculation in a loop, pausing to 50 milliseconds if th
   while(BarsCalculated(h)<=0)
     {
      k++;
      //--- Show the number of attempts
      PrintFormat("%s: k=%d",__FUNCTION__,k);
      //--- Wait 50 milliseconds to wait until the indicator is calculated
      Sleep(50);
      //--- If more than 100 attempt, then something is wrong
      if(k>100)
        {
         //--- Report a problem
         PrintFormat("Failed to calculate the indicator for %d attempts!")
         //--- Terminate the program operation
         return false;
        }
     }
//--- Everything is ready, the indicator is created and values are calcula
```

```
   return true;
  }
//+------------------------------------------------------------------+
//|  Searches for the last 2 zigzag fractures and places to arrays   |
//+------------------------------------------------------------------+
bool GetLastTwoFractures(double &get_values[],datetime &get_times[],int ha
  {
   double values[];         // An array for the values of the zigzag
   datetime times[];        // An array to get time
   int size=100;            // Size of the array
   ResetLastError();
//--- Copy the last 100 values of the indicator
   int copied=CopyBuffer(handle,0,0,size,values);
//--- Check the number of values copied
   if(copied<100)
     {
      PrintFormat("%s: Failed to copy %d values of the indicator with the
                  __FUNCTION__,size,handle,GetLastError());
      return false;
     }
//--- Define the order of access to the array as in a timeseries
   ArraySetAsSeries(values,true);
//--- Write here the numbers of bars, in which fractures were found
   int positions[];
//--- Set array sizes
   ArrayResize(get_values,3); ArrayResize(get_times,3); ArrayResize(positi
//--- Counters
   int i=0,k=0;
//--- Start to search for fractures
   while(i<100)
     {
      double v=values[i];
      //--- We are not interested in empty values
      if(v!=0.0)
        {
         //--- Remember the bar number
         positions[k]=i;
         //--- Remember the value of a zigzag on the fracture
         get_values[k]=values[i];
         PrintFormat("%s: Zigzag[%d]=%G",__FUNCTION__,i,values[i]);
         //--- Increase the counter
         k++;
         //--- If two fractures found, break the loop
         if(k>2) break;
        }
      i++;
     }
```

```
//--- Define the order of access to the arrays as in a timeseries
   ArraySetAsSeries(times,true);   ArraySetAsSeries(get_times,true);
   if(CopyTime(_Symbol,_Period,0,size,times)<=0)
     {
      PrintFormat("%s: Failed to copy %d values from CopyTime(). Error coc
                  __FUNCTION__,size,GetLastError());
      return false;
     }
//--- Open the bar open time, on which the last 2 fractures occurred
   get_times[0]=times[positions[1]];// The last but one value will be writ
   get_times[1]=times[positions[2]];// The value third from the end will b
   PrintFormat("%s: first=%s,  second=%s",__FUNCTION__,TimeToString(get_ti
//--- Successful
   return true;
  }
```

See also

[ChartApplyTemplate()](), [Resources]()

ChartWindowFind

The function returns the number of a subwindow where an indicator is drawn. There are 2 variants of the function.

1. The function searches in the indicated chart for the subwindow with the specified "short name" of the indicator (the short name is displayed in the left top part of the subwindow), and it returns the subwindow number in case of success.

```
int  ChartWindowFind(    long      chart_id,                        // chart iden
     string   indicator_shortname        // short indicator name, see INDICA
     );
```

2. The function must be called from a custom indicator. It returns the number of the subwindow where the indicator is working.

```
int  ChartWindowFind();
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

indicator_shortname

[in] Short name of the indicator.

Return Value

Subwindow number in case of success. In case of failure the function returns -1.

Note

If the second variant of the function (without parameters) is called from a script or Expert Advisor, the function returns -1.

Don't mix up the short name of an indicator and a file name, which is specified when an indicator is created using iCustom() function. If the indicator's short name is not set explicitly, then the name of the file containing the source code of the indicator, is specified in it during compilation.

It is important to correctly form the short name of an indicator, which is recorded in the INDICATOR_SHORTNAME property using the IndicatorSetString() function. It is recommended that the short name contains values of the indicator's input parameters, because the indicator deleted from a chart in the ChartIndicatorDelete() function is identified by its short name.

Example:

```
#property script_show_inputs
//--- input parameters
input string   shortname="MACD(12,26,9)";
//+------------------------------------------------------------------+
//| Returns number of the chart window with this indicator           |
//+------------------------------------------------------------------+
int GetIndicatorSubWindowNumber(long chartID=0,string short_name="")
  {
   int window=-1;
//---
   if((ENUM_PROGRAM_TYPE)MQLInfoInteger(MQL_PROGRAM_TYPE)==PROGRAM_INDICAT
     {
      //--- the function is called from the indicator, name is not require
      window=ChartWindowFind();
     }
   else
     {
      //--- the function is called from an Expert Advisor or script
      window=ChartWindowFind(0,short_name);
      if(window==-1) Print(__FUNCTION__+"(): Error = ",GetLastError());
     }
//---
   return(window);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int window=GetIndicatorSubWindowNumber(0,shortname);
   if(window!=-1)
      Print("Indicator "+shortname+" is in the window #"+(string)window);
   else
      Print("Indicator "+shortname+" is not found. window = "+(string)winc
  }
```

See also

ObjectCreate(), ObjectFind()

ChartTimePriceToXY

Converts the coordinates of a chart from the time/price representation to the X and Y coordinates.

```
bool  ChartTimePriceToXY(    long             chart_id,     // Chart ID
   int              sub_window,    // The number of the subwindow
   datetime         time,          // Time on the chart
   double           price,         // Price on the chart
   int&             x,             // The X coordinate for the time on the ch
   int&             y              // The Y coordinates for the price on the
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

sub_window

[in] The number of the chart subwindow. 0 means the main chart window.

time

[in] The time value on the chart, for which the value in pixels along the X axis will be received.

price

[in] The price value on the chart, for which the value in pixels along the Y axis will be received.

x

[out] The variable, into which the conversion of time to X will be received. The origin is in the upper left corner of the main chart window.

y

[out] The variable, into which the conversion of price to Y will be received. The origin is in the upper left corner of the main chart window.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the GetLastError() function.

See also

ChartXYToTimePrice()

ChartXYToTimePrice

Converts the X and Y coordinates on a chart to the time and price values.

```
bool  ChartXYToTimePrice(    long           chart_id,     // Chart ID
   int            x,            // The X coordinate on the chart
   int            y,            // The Y coordinate on the chart
   int&           sub_window,   // The number of the subwindow
   datetime&      time,         // Time on the chart
   double&        price         // Price on the chart
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

x

[in] The X coordinate. The origin is in the upper left corner of the main chart window.

y

[in] The Y coordinate. The origin is in the upper left corner of the main chart window.

sub_window

[out] The variable, into which the chart subwindow number will be written. 0 means the main chart window.

time

[out] The time value on the chart, for which the value in pixels along the X axis will be received.

price

[out] The price value on the chart, for which the value in pixels along the Y axis will be received.

Return Value

Returns true if successful, otherwise false. To get information about the error, call the GetLastError() function.

Example:

```
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- Show the event parameters on the chart
   Comment(__FUNCTION__,": id=",id," lparam=",lparam," dparam=",dparam," s
//--- If this is an event of a mouse click on the chart
   if(id==CHARTEVENT_CLICK)
     {
      //--- Prepare variables
      int      x     =(int)lparam;
      int      y     =(int)dparam;
      datetime dt    =0;
      double   price =0;
      int      window=0;
      //--- Convert the X and Y coordinates in terms of date/time
      if(ChartXYToTimePrice(0,x,y,window,dt,price))
        {
         PrintFormat("Window=%d X=%d  Y=%d  =>  Time=%s  Price=%G",window,
         //--- Perform reverse conversion: (X,Y) => (Time,Price)
         if(ChartTimePriceToXY(0,window,dt,price,x,y))
            PrintFormat("Time=%s  Price=%G  =>  X=%d  Y=%d",TimeToString(c
         else
            Print("ChartTimePriceToXY return error code: ",GetLastError())
         //--- delete lines
         ObjectDelete(0,"V Line");
         ObjectDelete(0,"H Line");
         //--- create horizontal and vertical lines of the crosshair
         ObjectCreate(0,"H Line",OBJ_HLINE,window,dt,price);
         ObjectCreate(0,"V Line",OBJ_VLINE,window,dt,price);
         ChartRedraw(0);
        }
      else
         Print("ChartXYToTimePrice return error code: ",GetLastError());
      Print("+--------------------------------------------------------------
     }
  }
```

See also

ChartTimePriceToXY()

ChartOpen

Opens a new chart with the specified symbol and period. The command is added to chart message queue and executed only after all previous commands have been processed.

```
long  ChartOpen(    string              symbol,      // Symbol name
   ENUM_TIMEFRAMES  period      // Period
   );
```

Parameters

symbol

[in] Chart symbol. NULL means the symbol of the current chart (the Expert Advisor is attached to).

period

[in] Chart period (timeframe). Can be one of the ENUM_TIMEFRAMES values. 0 means the current chart period.

Return Value

If successful, it returns the opened chart ID. Otherwise returns 0. To get error details use the GetLastError() function.

Note

The maximum possible number of simultaneously open charts in the terminal can't exceed the CHARTS_MAX value.

ChartFirst

Returns the ID of the first chart of the client terminal.

```
long  ChartFirst();
```

Return Value

Chart ID.

ChartNext

Returns the chart ID of the chart next to the specified one.

```
long  ChartNext(    long  chart_id       // Chart ID
   );
```

Parameters

chart_id

[in] Chart ID. 0 does not mean the current chart. 0 means "return the first chart ID".

Return Value

Chart ID. If this is the end of the chart list, it returns -1.

Example:

```
//--- variables for chart ID
   long currChart,prevChart=ChartFirst();
   int i=0,limit=100;
   Print("ChartFirst =",ChartSymbol(prevChart)," ID =",prevChart);
   while(i<limit)// We have certainly not more than 100 open charts
     {
      currChart=ChartNext(prevChart); // Get the new chart ID by using the
      if(currChart<0) break;          // Have reached the end of the chart
      Print(i,ChartSymbol(currChart)," ID =",currChart);
      prevChart=currChart;// let's save the current chart ID for the Chart
      i++;// Do not forget to increase the counter
     }
```

ChartClose

Closes the specified chart.

```
bool  ChartClose(    long  chart_id=0       // Chart ID
   );
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Return Value

If successful, returns true, otherwise false.

ChartSymbol

Returns the symbol name for the specified chart.

```
string  ChartSymbol(    long   chart_id=0       // Chart ID
   );
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Return Value

If chart does not exist, the result will be an empty string.

See also

ChartSetSymbolPeriod()

ChartPeriod

Returns the timeframe period of specified chart.

```
ENUM_TIMEFRAMES  ChartPeriod(    long  chart_id=0       // Chart ID
   );
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Return Value

The function returns one of the ENUM_TIMEFRAMES values. If chart does not exist, it returns 0.

ChartRedraw

This function calls a forced redrawing of a specified chart.

```
void  ChartRedraw(    long  chart_id=0       // Chart ID
   );
```

Parameters

chart_id=0

[in] Chart ID. 0 means the current chart.

Note

Usually it is used after changing the object properties.

When the ChartRedraw() function is called from an indicator, the chart is redrawn only after the calculation of the OnCalculate() function is over, because indicators are executed in the common terminal thread.

See also

Objects functions

ChartSetDouble

Sets a value for a corresponding property of the specified chart. Chart property should be of a double type. The command is added to chart message queue and executed only after all previous commands have been processed.

```
bool  ChartSetDouble(   long    chart_id,     // Chart ID
   int     prop_id,      // Property ID
   double  value         // Value
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. Can be one of the ENUM_CHART_PROPERTY_DOUBLE values (except the read-only properties).

value

[in] Property value.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get error details use the GetLastError() function.

ChartSetInteger

Sets a value for a corresponding property of the specified chart. Chart property must be datetime, int, color, bool or char. The command is added to chart message queue and executed only after all previous commands have been processed.

```
bool  ChartSetInteger(   long   chart_id,     // Chart ID
   int     prop_id,      // Property ID
   long    value         // Value
   );
```

Sets a value for a corresponding property of the specified subwindow of the specified chart:

```
bool  ChartSetInteger(
   long    chart_id,     // Chart ID
   int     property_id,  // Property ID
   uint    sub_window,   // Chart subwindow
   long    value         // Value
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. It can be one of the ENUM_CHART_PROPERTY_INTEGER value (except the read-only properties).

sub_window

[in] Chart subwindow.

value

[in] Property value.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get error details use the GetLastError() function.

ChartSetString

Sets a value for a corresponding property of the specified chart. Chart property must be of the string type. The command is added to chart message queue and executed only after all previous commands have been processed.

```
bool  ChartSetString(    long     chart_id,       // Chart ID
   int      prop_id,       // Property ID
   string   str_value      // Value
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. Its value can be one of the ENUM_CHART_PROPERTY_STRING values (except the read-only properties).

str_value

[in] Property value string. String length cannot exceed 2045 characters (extra characters will be truncated).

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get error details use the GetLastError() function.

Note

ChartSetString can be used for a comment output on the chart instead of the Comment function.

Example:

```
void OnTick()
  {
//---
   double Ask,Bid;
   int Spread;
   Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
   string comment=StringFormat("Printing prices:\nAsk = %G\nBid = %G\nSpre
                               Ask,Bid,Spread);
   ChartSetString(0,CHART_COMMENT,comment);
  }
```

See also

Comment, ChartGetString

ChartGetDouble

Returns the value of a corresponding property of the specified chart. Chart property must be of double type. There are 2 variants of the function calls.

1. Returns the property value directly.

```
double  ChartGetDouble(    long   chart_id,          // Chart ID
   int    prop_id,            // Property ID
   int    sub_window=0        // subwindow number, if necessary
   );
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable double_var passed by reference.

```
bool  ChartGetDouble(
   long     chart_id,         // Chart ID
   int      prop_id,          // Property ID
   int      sub_window,       // Subwindow number
   double&  double_var        // Target variable for the chart property
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. This value can be one of the ENUM_CHART_PROPERTY_DOUBLE values.

sub_window

[in] Number of the chart subwindow. For the first case, the default value is 0 (main chart window). The most of the properties do not require a subwindow number.

double_var

[out] Target variable of double type for the requested property.

Return Value

The value of double type.

For the second call case it returns true if the specified property is available and its value has been placed into double_var variable, otherwise returns false. To get an additional information about the error, it is necessary to call the function GetLastError().

Example:

```
void OnStart()
  {
   double priceMin=ChartGetDouble(0,CHART_PRICE_MIN,0);
   double priceMax=ChartGetDouble(0,CHART_PRICE_MAX,0);
   Print("CHART_PRICE_MIN =",priceMin);
   Print("CHART_PRICE_MAX =",priceMax);
  }
```

ChartGetInteger

Returns the value of a corresponding property of the specified chart. Chart property must be of datetime, int or bool type. There are 2 variants of the function calls.

1. Returns the property value directly.

```
long  ChartGetInteger(    long  chart_id,          // Chart ID
   int    prop_id,            // Property ID
   int    sub_window=0        // subwindow number, if necessary
   );
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable long_var passed by reference.

```
bool  ChartGetInteger(
   long     chart_id,         // Chart ID
   int      prop_id,          // Property ID
   int      sub_window,       // subwindow number
   long&    long_var          // Target variable for the property
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. This value can be one of the ENUM_CHART_PROPERTY_INTEGER values.

sub_window

[in] Number of the chart subwindow. For the first case, the default value is 0 (main chart window). The most of the properties do not require a subwindow number.

long_var

[out] Target variable of long type for the requested property.

Return Value

The value of long type.

For the second call case it returns true if specified property is available and its value has been stored into long_var variable, otherwise returns false. To get additional information about the error, it is necessary to call the function GetLastError().

Example:

```
void OnStart()
  {
   int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
   int width=ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0);
   Print("CHART_HEIGHT_IN_PIXELS =",height,"pixels");
   Print("CHART_WIDTH_IN_PIXELS =",width,"pixels");
  }
```

ChartGetString

Returns the value of a corresponding property of the specified chart. Chart property must be of string type. There are 2 variants of the function call.

1. Returns the property value directly.

```
string  ChartGetString(    long   chart_id,          // Chart ID
   int    prop_id             // Property ID
   );
```

2. Returns true or false, depending on the success of a function. If successful, the value of the property is placed in a target variable string_var passed by reference.

```
bool  ChartGetString(
   long     chart_id,         // Chart ID
   int      prop_id,          // Property ID
   string&  string_var        // Target variable for the property
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

prop_id

[in] Chart property ID. This value can be one of the ENUM_CHART_PROPERTY_STRING values.

string_var

[out] Target variable of string type for the requested property.

Return Value

The value of string type.

For the second call case it returns true if the specified property is available and its value has been stored into string_var variable, otherwise returns false. To get additional information about the error, it is necessary to call the function GetLastError().

Note

ChartGetString can be used for reading comments plotted on the chart using the Comment or ChartSetString functions.

Example:

```
void OnStart()
  {
   ChartSetString(0,CHART_COMMENT,"Test comment.\nSecond line.\nThird!");
   ChartRedraw();
   Sleep(1000);
   string comm=ChartGetString(0,CHART_COMMENT);
   Print(comm);
  }
```

See also

Comment, ChartSetString

ChartNavigate

Performs shift of the specified chart by the specified number of bars relative to the specified position in the chart. The command is added to chart message queue and executed only after all previous commands have been processed.

```
bool  ChartNavigate(    long  chart_id,      // Chart ID
   int    position,      // Position
   int    shift=0        // Shift value
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

position

[in] Chart position to perform a shift. Can be one of the ENUM_CHART_POSITION values.

shift=0

[in] Number of bars to shift the chart. Positive value means the right shift (to the end of chart), negative value means the left shift (to the beginning of chart). The zero shift can be used to navigate to the beginning or end of chart.

Return Value

Returns true if successful, otherwise returns false.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get handle of the current chart
   long handle=ChartID();
   string comm="";
   if(handle>0) // if successful, additionally set up the chart
     {
      //--- disable auto scroll
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      //--- set a shift from the right chart border
      ChartSetInteger(handle,CHART_SHIFT,true);
      //--- draw candlesticks
      ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
      //--- set the display mode for tick volumes
      ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);

      //--- prepare a text to output in Comment()
      comm="Scroll 10 bars to the right of the history start";
      //--- show comment
      Comment(comm);
      //--- scroll 10 bars to the right of the history start
      ChartNavigate(handle,CHART_BEGIN,10);
      //--- get the number of the first bar visible on the chart (numerati
      long first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
      //--- add line feed character
      comm=comm+"\r\n";
      //--- add to comment
      comm=comm+"The first bar on the chart is number "+IntegerToString(fi
      //--- show comment
      Comment(comm);
      //--- wait 5 seconds to see how the chart moves
      Sleep(5000);

      //--- add to the comment text
      comm=comm+"\r\n"+"Scroll 10 bars to the left of the right chart bord
      Comment(comm);
      //--- scroll 10 bars to the left of the right chart border
      ChartNavigate(handle,CHART_END,-10);
      //--- get the number of the first bar visible on the chart (numerati
      first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
      comm=comm+"\r\n";
      comm=comm+"The first bar on the chart is number "+IntegerToString(fi
      Comment(comm);
      //--- wait 5 seconds to see how the chart moves
      Sleep(5000);

      //--- new block of chart scrolling
      comm=comm+"\r\n"+"Scroll 300 bars to the right of the history start"
      Comment(comm);
      //--- scroll 300 bars to the right of the history start
      ChartNavigate(handle,CHART_BEGIN,300);
      first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
      comm=comm+"\r\n";
      comm=comm+"The first bar on the chart is number "+IntegerToString(fi
      Comment(comm);
      //--- wait 5 seconds to see how the chart moves
      Sleep(5000);

      //--- new block of chart scrolling
```

```
      comm=comm+"\r\n"+"Scroll 300 bars to the left of the right chart bor
      Comment(comm);
      //--- scroll 300 bars to the left of the right chart border
      ChartNavigate(handle,CHART_END,-300);
      first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
      comm=comm+"\r\n";
      comm=comm+"The first bar on the chart is number "+IntegerToString(fi
      Comment(comm);
     }
  }
```

ChartID

Returns the ID of the current chart.

```
long  ChartID();
```

Return Value

Value of long type.

ChartIndicatorDelete

Removes an indicator with a specified name from the specified chart window. The command is added to chart message queue and executed only after all previous commands have been processed.

```
bool  ChartIndicatorDelete(    long           chart_id,              // ch
   int            sub_window,             // number of the subwindow
   const string   indicator_shortname     // short name of the indicator
   );
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

sub_window

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

const indicator_shortname

[in] The short name of the indicator which is set in the INDICATOR_SHORTNAME property with the IndicatorSetString() function. To get the short name of an indicator use the ChartIndicatorName() function.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get error details use the GetLastError() function.

Note

If two indicators with identical short names exist in the chart subwindow, the first one in a row will be deleted.

If other indicators on this chart are based on the values of the indicator that is being deleted, such indicators will also be deleted.

If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

The indicator's short name should be formed correctly. It will be written to the INDICATOR_SHORTNAME property using the IndicatorSetString() function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the ChartIndicatorDelete() function is identified by the short name.

Example of deleting an indicator after initialization has failed:

```
//+------------------------------------------------------------------+
//|                                    Demo_ChartIndicatorDelete.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 1
//--- plot Histogram
#property indicator_label1  "Histogram"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      first_param=1;
input int      second_param=2;
input int      third_param=3;
input bool     wrong_init=true;
//--- indicator buffers
double         HistogramBuffer[];
string         shortname;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
   int res=INIT_SUCCEEDED;
//--- Link the HistogramBuffer array to the indicator buffer
   SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- Construct a short indicator name based on input parameters
   shortname=StringFormat("Demo_ChartIndicatorDelete(%d,%d,%d)",
                          first_param,second_param,third_param);
   IndicatorSetString(INDICATOR_SHORTNAME,shortname);
//--- If forced completion of an indicator is set, return a non-zero value
   if(wrong_init) res=INIT_FAILED;
   return(res);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//--- Starting position for working in a loop
   int start=prev_calculated-1;
   if(start<0) start=0;
//--- Fill in the indicator buffer with values
   for(int i=start;i<rates_total;i++)
     {
      HistogramBuffer[i]=close[i];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Handler of the Deinit event                                      |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   PrintFormat("%s: Deinitialization reason code=%d",__FUNCTION__,reason);
   if(reason==REASON_INITFAILED)
     {
      PrintFormat("An indicator with a short name %s (file %s) deletes its
      int window=ChartWindowFind();
      bool res=ChartIndicatorDelete(0,window,shortname);
      //--- Analyse the result of call of ChartIndicatorDelete()
      if(!res)
        {
         PrintFormat("Failed to delete indicator %s from window #%d. Error
                     shortname,window,GetLastError());
        }
     }
  }
```

See also

ChartIndicatorName(), ChartIndicatorsTotal(), IndicatorSetString()

ChartIndicatorName

Returns the short name of the indicator by the number in the indicators list on the specified chart window.

```
string  ChartIndicatorName(    long   chart_id,       // chart id
   int    sub_window,     // number of the subwindow
   int    index           // index of the indicator in the list of indicator
   );
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

sub_window

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

index

[in] the index of the indicator in the list of indicators. The numeration of indicators start with zero, i.e. the first indicator in the list has the 0 index. To obtain the number of indicators in the list use the ChartIndicatorsTotal() function.

Return Value

The short name of the indicator which is set in the INDICATOR_SHORTNAME property with the IndicatorSetString() function. To get error details use the GetLastError() function.

Note

If the short name of an indicator is not set explicitly, then the name of the file containing the source code of the indicator will be specified during compilation.

The indicator's short name should be formed correctly. It will be written to the INDICATOR_SHORTNAME property using the IndicatorSetString() function. It is recommended that the short name should contain values of all the input parameters of the indicator, because the indicator to be deleted from the chart by the ChartIndicatorDelete() function is identified by the short name.

See also

ChartIndicatorDelete(), ChartIndicatorsTotal(), IndicatorSetString()

ChartIndicatorsTotal

Returns the number of all indicators applied to the specified chart window.

```
int  ChartIndicatorsTotal(    long  chart_id,       // chart id
   int    sub_window      // number of the subwindow
   );
```

Parameters

chart_id

[in] Chart ID. 0 denotes the current chart.

sub_window

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

Return Value

The number of indicators in the specified chart window. To get error details use the GetLastError() function.

Note

The function allows going searching through all the indicators attached to the chart. The number of all the windows of the chart can be obtained from the CHART_WINDOWS_TOTAL property using the ChartGetInteger() function.

See also

ChartIndicatorDelete(), ChartIndicatorsTotal(), IndicatorSetString()

ChartWindowOnDropped

Returns the number (index) of the chart subwindow the Expert Advisor or script has been dropped to. 0 means the main chart window.

```
int  ChartWindowOnDropped();
```

Return Value

Value of int type.

Example:

```
   int myWindow=ChartWindowOnDropped();    int windowsTotal=ChartGetIntege
   Print("Script is running on the window #"+myWindow+
         ". Total windows on the chart "+ChartSymbol()+":",windowsTotal);
```

See also

ChartPriceOnDropped(), ChartTimeOnDropped(), ChartXOnDropped(), ChartYOnDropped()

ChartPriceOnDropped

Returns the price coordinate corresponding to the chart point the Expert Advisor or script has been dropped to.

```
double  ChartPriceOnDropped();
```

Return Value

Value of double type.

Example:

```
   double p=ChartPriceOnDropped();    Print("ChartPriceOnDropped() = ",p);
```

See also

ChartXOnDropped(), ChartYOnDropped()

ChartTimeOnDropped

Returns the time coordinate corresponding to the chart point the Expert Advisor or script has been dropped to.

```
datetime  ChartTimeOnDropped();
```

Return Value

Value of datetime type.

Example:

```
   datetime t=ChartTimeOnDropped();    Print("Script was dropped on the "+
```

See also

ChartXOnDropped(), ChartYOnDropped()

ChartXOnDropped

Returns the X coordinate of the chart point the Expert Advisor or script has been dropped to.

```
int  ChartXOnDropped();
```

Return Value

The X coordinate value.

Note

X axis direction from left to right.

Example:

```
   int X=ChartXOnDropped();    int Y=ChartYOnDropped();
   Print("(X,Y) = ("+X+","+Y+")");
```

See also

[ChartWindowOnDropped()](), [ChartPriceOnDropped()](), [ChartTimeOnDropped()]()

ChartYOnDropped

Returns the Y coordinateof the chart point the Expert Advisor or script has been dropped to.

```
int  ChartYOnDropped();
```

Return Value

The Y coordinate value.

Note

Y axis direction from top to bottom.

See also

ChartWindowOnDropped(), ChartPriceOnDropped(), ChartTimeOnDropped()

ChartSetSymbolPeriod

Changes the symbol and period of the specified chart. The function is asynchronous, i.e. it sends the command and does not wait for its execution completion. The command is added to chart message queue and executed only after all previous commands have been processed.

```
bool  ChartSetSymbolPeriod(    long             chart_id,      // Chart ID
   string           symbol,        // Symbol name
   ENUM_TIMEFRAMES  period         // Period
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

symbol

[in] Chart symbol. NULL value means the current chart symbol (Expert Advisor is attached to)

period

[in] Chart period (timeframe). Can be one of the ENUM_TIMEFRAMES values. 0 means the current chart period.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get error details use the GetLastError() function.

Note

The symbol/period change leads to the re-initialization of the Expert Advisor, attached to a chart. Re-initialization is not performed on offline charts, they're only refreshed (the same as when clicking Refresh in the terminal).

See also

ChartSymbol(), ChartPeriod()

ChartScreenShot

Saves current chart screen shot as a GIF, PNG or BMP file depending on specified extension. The command is added to chart message queue and executed only after all previous commands have been processed.

```
bool  ChartScreenShot(    long              chart_id,                 //
   string              filename,                  // Symbol name
   int                 width,                     // Width
   int                 height,                    // Height
   ENUM_ALIGN_MODE     align_mode=ALIGN_RIGHT     // Alignment type
   );
```

Parameters

chart_id

[in] Chart ID. 0 means the current chart.

filename

[in] Screenshot file name. Cannot exceed 63 characters. Screenshot files are placed in the \Files directory.

width

[in] Screenshot width in pixels.

height

[in] Screenshot height in pixels.

align_mode=ALIGN_RIGHT

[in] Output mode of a narrow screenshot. A value of the ENUM_ALIGN_MODE enumeration. ALIGN_RIGHT means align to the right margin (the output from the end). ALIGN_LEFT means Left justify.

Return Value

Returns true if the command has been added to chart queue, otherwise false. To get error details use the GetLastError() function.

Note

If you need to take a screenshot from a chart from a certain position, first it's necessary to position the graph using the ChartNavigate() function. If the horizontal size of the screenshot is smaller than the chart window, either the right part of the chart window, or its left part is output, depending on the align_mode settings.

Example:

```
#property description "The Expert Advisor demonstrates how to create a ser
#property description "chart using the ChartScreenShot() function. For cor
#property description "shown on the chart. The height and width of images
//---
#define        WIDTH  800     // Image width to call ChartScreenShot()
#define        HEIGHT 600     // Image height to call ChartScreenShot()
//--- input parameters
input int      pictures=5;    // The number of images in the series
int            mode=-1;       // -1 denotes a shift to the right edge of t
int            bars_shift=300;// The number of bars when scrolling the cha
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- Disable chart autoscroll
   ChartSetInteger(0,CHART_AUTOSCROLL,false);
//--- Set the shift of the right edge of the chart
   ChartSetInteger(0,CHART_SHIFT,true);
//--- Show a candlestick chart
   ChartSetInteger(0,CHART_MODE,CHART_CANDLES);
//---
   Print("Preparation of the Expert Advisor is completed");
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- Show the name of the function, call time and event identifier
   Print(__FUNCTION__,TimeCurrent(),"   id=",id,"   mode=",mode);
//--- Handle the CHARTEVENT_CLICK event ("A mouse click on the chart")
   if(id==CHARTEVENT_CLICK)
     {
      //--- Initial shift from the chart edge
      int pos=0;
      //--- Operation with the left chart edge
      if(mode>0)
        {
         //--- Scroll the chart to the left edge
         ChartNavigate(0,CHART_BEGIN,pos);
         for(int i=0;i<pictures;i++)
           {
            //--- Prepare a text to show on the chart and a file name
            string name="ChartScreenShot"+"CHART_BEGIN"+string(pos)+".gif"
            //--- Show the name on the chart as a comment
            Comment(name);
            //--- Save the chart screenshot in a file in the terminal_dire
            if(ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_LEFT))
               Print("We've saved the screenshot ",name);
            //---
            pos+=bars_shift;
            //--- Give the user time to look at the new part of the chart
            Sleep(3000);
            //--- Scroll the chart from the current position bars_shift ba
            ChartNavigate(0,CHART_CURRENT_POS,bars_shift);
           }
         //--- Change the mode to the opposite
         mode*=-1;
        }
      else // Operation with the right chart edge
        {
         //--- Scroll the chart to the right edge
         ChartNavigate(0,CHART_END,pos);
         for(int i=0;i<pictures;i++)
           {
            //--- Prepare a text to show on the chart and a file name
            string name="ChartScreenShot"+"CHART_END"+string(pos)+".gif";
            //--- Show the name on the chart as a comment
            Comment(name);
            //--- Save the chart screenshot in a file in the terminal_dire
            if(ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_RIGHT))
               Print("We've saved the screenshot ",name);
            //---
            pos+=bars_shift;
            //--- Give the user time to look at the new part of the chart
            Sleep(3000);
            //--- Scroll the chart from the current position bars_shift ba
            ChartNavigate(0,CHART_CURRENT_POS,-bars_shift);
           }
         //--- Change the mode to the opposite
         mode*=-1;
        }
     }  // End of CHARTEVENT_CLICK event handling
```

```
//--- End of the OnChartEvent() handler
  }
```

See also

ChartNavigate(), Resources

Period

Returns timeframe of the current chart.

```
int  Period();
```

Parameters

None.

Returned value

Period (timeframe) of the current chart (in minutes).

Example:

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

Symbol

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

```
string  Symbol();
```

Parameters

None.

Returned value

A text string with the name of the current financial instrument.

Example:

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

WindowBarsPerChart

Returns the amount of bars visible on the chart.

```
int  WindowBarsPerChart();
```

Parameters

None.

Returned value

The amount of bars visible on the chart.

Example:

```
// work with visible bars.   int bars_count=WindowBarsPerChart();
  int bar=WindowFirstVisibleBar();
  for(int i=0; i<bars_count; i++,bar--)
    {
     // ...
    }
```

WindowExpertName

Returns the name of the executed Expert Advisor, script, custom indicator, or library.

```
string  WindowExpertName();
```

Parameters

None.

Returned value

The name of the executed Expert Advisor, script, custom indicator, or library, depending on the MQL4 program, from which this function has been called.

Example:

```
  string name=WindowExpertName();   GlobalVariablesDeleteAll(name);
```

WindowFind

Returns the window index containing this specified indicator.

```
int  WindowFind(    string        name    // name
   );
```

Parameters

name

[in] Indicator short name.

Returned value

If indicator with name was found, the function returns the window index containing this specified indicator, otherwise it returns -1.

Note

WindowFind() returns -1 if custom indicator searches itself when init() function works.

Example:

```
  int win_idx=WindowFind("MACD(12,26,9)");
```

WindowFirstVisibleBar

Returns index of the first visible bar in the current chart window.

```
int  WindowFirstVisibleBar();
```

Parameters

None.

Returned value

Index of the first visible bar number in the current chart window.

Note

It must be taken into consideration that price bars are numbered in the reverse order, from the last to the first one. The current bar, the latest in the price array, is indexed as 0. The oldest bar is indexed as Bars-1. If the first visible bar number is 2 or more bars less than the amount of visible bars in the chart, it means that the chart window has not been fully filled out and there is a space to the left.

Example:

```
   // work with visible bars.    int bars_count=WindowBarsPerChart();
   int bar=WindowFirstVisibleBar();
   for(int i=0; i<bars_count; i++,bar--)
     {
      // ...
     }
```

WindowHandle

Returns the system handle of the chart window.

```
int  WindowHandle(    string         symbol,      // symbol
   int            timeframe    // timeframe
   );
```

Parameters

symbol

[in] Symbol.

timeframe

[in] Timeframe. It can be any of Timeframe enumeration values. 0 means the current chart timeframe.

Returned value

Returns the system handle of the chart window. If the chart of symbol and timeframe has not been opened by the moment of function calling, 0 will be returned.

Example:

```
  int win_handle=WindowHandle("USDX",PERIOD_H1);
  if(win_handle!=0)
    Print("Window with USDX,H1 detected. Rates array will be copied immedi
```

WindowIsVisible

Returns the visibility flag of the chart subwindow.

```
int  WindowIsVisible(    int          index      // subwindow
   );
```

Parameters

index

[in] Subwindow index.

Returned value

Returns true if the chart subwindow is visible, otherwise returns false. The chart subwindow can be hidden due to the visibility properties of the indicator placed in it.

Example:

```
  int maywin=WindowFind("MyMACD");
  if(maywin>-1 && WindowIsVisible(maywin)==true)
    Print("window of MyMACD is visible");
  else
    Print("window of MyMACD not found or is not visible");
```

WindowOnDropped

Returns the window index where Expert Advisor, custom indicator or script was dropped.

```
int  WindowOnDropped();
```

Parameters

None.

Returned value

The window index where Expert Advisor, custom indicator or script was dropped. This value is valid if the Expert Advisor, custom indicator or script was dropped by mouse.

Note

For custom indicators being initialized (call from the init() function), this index is not defined.

The returned index is the number of window (0-chart main menu, subwindows of indicators are numbered starting from 1) where the custom indicator is working. A custom indicator can create its own new subwindow during its work, and the number of this subwindow will differ from that of the window where the indicator was really dropped in.

Example:

```
  if(WindowOnDropped()!=0)     {
     Print("Indicator 'MyIndicator' must be applied to main chart window!"
     return(false);
    }
```

See also

[WindowXOnDropped()](), [WindowYOnDropped()]()

WindowPriceMax

Returns the maximal value of the vertical scale of the specified subwindow of the current chart.

```
int  WindowPriceMax(    int         index=0    // subwindow
   );
```

Parameters

index=0

[in] Chart subwindow index (0 - main chart window).

Returned value

The maximal value of the vertical scale of the specified subwindow of the current chart (0-main chart window, the indicators' subwindows are numbered starting from 1). If the subwindow index has not been specified, the maximal value of the price scale of the main chart window is returned.

Example:

```
  double   top=WindowPriceMax();
  double   bottom=WindowPriceMin();
  datetime left=Time[WindowFirstVisibleBar()];
  int      right_bound=WindowFirstVisibleBar()-WindowBarsPerChart();
  if(right_bound<0) right_bound=0;
  datetime right=Time[right_bound]+Period()*60;
//----
  ObjectCreate("Padding_rect",OBJ_RECTANGLE,0,left,top,right,bottom);
  ObjectSet("Padding_rect",OBJPROP_BACK,true);
  ObjectSet("Padding_rect",OBJPROP_COLOR,Blue);
  WindowRedraw();
```

See also

WindowPriceMin(), WindowFirstVisibleBar(), WindowBarsPerChart()

WindowPriceMin

Returns the minimal value of the vertical scale of the specified subwindow of the current chart.

```
int  WindowPriceMin(    int          index=0    // subwindow
   );
```

Parameters

index=0

[in] Chart subwindow index (0 - main chart window).

Returned value

The minimal value of the vertical scale of the specified subwindow of the current chart (0-main chart window, the indicators' subwindows are numbered starting from 1). If the subwindow index has not been specified, the minimal value of the price scale of the main chart window is returned.

Example:

```
  double    top=WindowPriceMax();
  double    bottom=WindowPriceMin();
  datetime left=Time[WindowFirstVisibleBar()];
  int       right_bound=WindowFirstVisibleBar()-WindowBarsPerChart();
  if(right_bound<0) right_bound=0;
  datetime right=Time[right_bound]+Period()*60;
//----
  ObjectCreate("Padding_rect",OBJ_RECTANGLE,0,left,top,right,bottom);
  ObjectSet("Padding_rect",OBJPROP_BACK,true);
  ObjectSet("Padding_rect",OBJPROP_COLOR,Blue);
  WindowRedraw();
```

See also

WindowPriceMax(), WindowFirstVisibleBar(), WindowBarsPerChart()

WindowPriceOnDropped

Returns the price of the chart point where Expert Advisor or script was dropped.

```
double  WindowPriceOnDropped();
```

Parameters

None.

Returned value

The price of the chart point where Expert Advisor or script was dropped. This value is only valid if the expert or script was dropped by mouse.

Note

For custom indicators this value is undefined.

Example:

```
  double   drop_price=WindowPriceOnDropped();   datetime drop_time=WindowT
  //---- 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);
    }
```

See also

WindowTimeOnDropped(), WindowYOnDropped(), WindowOnDropped()

WindowRedraw

Redraws the current chart forcedly.

```
void  WindowRedraw();
```

Parameters

None.

Returned value

None.

Note

Redraws the current chart forcedly. It is normally used after the objects properties have been changed.

Example:

```
  //---- set new properties for some objects   ObjectMove(object_name1, 0,
  ObjectSet(object_name1, OBJPROP_ANGLE, angle*2);
  ObjectSet(object_name1, OBJPROP_FONTSIZE, fontsize);
  ObjectSet(line_name, OBJPROP_TIME2, time2);
  ObjectSet(line_name, OBJPROP_ANGLE, line_angle);
  //---- now redraw all
  WindowRedraw();
```

WindowScreenShot

Saves current chart screen shot as a GIF file.

```
bool  WindowScreenShot(    string              filename,             //
   int                 size_x,                    // width
   int                 size_y,                    // height
   int                 start_bar=-1,              // first visible bar
   int                 chart_scale=-1,            // scale
   int                 chart_mode=-1              // mode
   );
```

Parameters

filename

[in] Screen shot file name. Screenshot is saved to \Files folder.

size_x

[in] Screen shot width in pixels.

size_y

[in] Screen shot height in pixels.

start_bar=-1

[in] Index of the first visible bar in the screen shot. If 0 value is set, the current first visible bar will be shot. If no value or negative value has been set, the end-of-chart screen shot will be produced, indent being taken into consideration.

chart_scale=-1

[in] Horizontal chart scale for screen shot. Can be in the range from 0 to 5. If no value or negative value has been set, the current chart scale will be used.

chart_mode=-1

[in] Chart displaying mode. It can take the following values: CHART_BAR (0 is a sequence of bars), CHART_CANDLE (1 is a sequence of candlesticks), CHART_LINE (2 is a close prices line). If no value or negative value has been set, the chart will be shown in its current mode.

Returned value

Returns true if succeed, otherwise false. To get the error code, one has to use the GetLastError() function.

Note

The screen shot is saved in the terminal_dir\experts\files (terminal_dir\tester\files in case of testing) directory or its subdirectories.

Example:

```
int lasterror=0;
//---- tester has closed one or more trades
if(IsTesting() && ExtTradesCounter<TradesTotal())
  {
   //---- make WindowScreenShot for further checking
   if(!WindowScreenShot("shots\\tester"+ExtShotsCounter+".gif",640,480))
      lasterror=GetLastError();
   else ExtShotsCounter++;
   ExtTradesCounter=TradesTotal();
  }
```

WindowTimeOnDropped

Returns the time of the chart point where Expert Advisor or script was dropped.

```
datetime  WindowTimeOnDropped();
```

Parameters

None.

Returned value

The time value of the chart point where expert or script was dropped. This value is only valid if the expert or script was dropped by mouse.

Notwe

For custom indicators this value is undefined.

Example:

```
   double   drop_price=WindowPriceOnDropped();   datetime drop_time=WindowT
   //---- 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);
     }
```

See also

WindowPriceOnDropped(), WindowYOnDropped(), WindowOnDropped()

WindowsTotal

Returns total number of indicator windows on the chart.

```
int  WindowsTotal();
```

Parameters

None.

Returned value

Total number of indicator windows on the chart (including main chart).

Example:

```
  Print("Total windows = ", WindowsTotal());
```

WindowXOnDropped

Returns the value at X axis in pixels for the chart window client area point at which the Expert Advisor or script was dropped.

```
int  WindowXOnDropped();
```

Parameters

None.

Returned value

The value at X axis in pixels for the chart window client area point at which the expert or script was dropped. The value will be true only if the expert or script were moved with the mouse ("Drag'n'Drop") technique.

Example:

```
   Print("Expert dropped at point x=",WindowXOnDropped()," y=",WindowYOnDro
```

See also

WindowYOnDropped(), WindowTimeOnDropped(), WindowOnDropped()

WindowYOnDropped

Returns the value at Y axis in pixels for the chart window client area point at which the Expert Advisor or script was dropped.

```
int  WindowYOnDropped();
```

Parameters

None.

Returned value

Returns the value at Y axis in pixels for the chart window client area point at which the Expert Advisor or script was dropped. The value will be true only if the expert or script were moved with the mouse ("Drag'n'Drop") technique.

Example:

```
   Print"Expert was attached to the window in the point x=",WindowXOnDroppe
```

See also

WindowYOnDropped(), WindowPriceOnDropped(), WindowOnDropped()

Trade Functions

This is the group of functions intended for managing trading activities.

Trading functions can be used in Expert Advisors and scripts. OrderSend(), OrderClose(), OrderCloseBy(), OrderModify(), OrderDelete() trading functions changing the state of a trading account can be called only if trading by Expert Advisors is allowed (the "Allow live trading" checkbox is enabled in the Expert Advisor or script properties).

Trading can be allowed or prohibited depending on various factors described in the Trade Permission section.

Function	Action
OrderClose	Closes opened order
OrderCloseBy	Closes an opened order by another opposite opened order
OrderClosePrice	Returns close price of the currently selected order
OrderCloseTime	Returns close time of the currently selected order
OrderComment	Returns comment of the currently selected order
OrderCommission	Returns calculated commission of the currently selected order
OrderDelete	Deletes previously opened pending order
OrderExpiration	Returns expiration date of the selected pending order
OrderLots	Returns amount of lots of the selected order
OrderMagicNumber	Returns an identifying (magic) number of the currently selected order
OrderModify	Modification of characteristics of the previously opened or pending orders
OrderOpenPrice	Returns open price of the currently selected order
OrderOpenTime	Returns open time of the currently selected order
OrderPrint	Prints information about the selected order in the log
OrderProfit	Returns profit of the currently selected order
OrderSelect	The function selects an order for further processing
OrderSend	The main function used to open an order or place a pending order
OrdersHistoryTotal	Returns the number of closed orders in the account history loaded into the terminal

OrderStopLoss	Returns stop loss value of the currently selected order
OrdersTotal	Returns the number of market and pending orders
OrderSwap	Returns swap value of the currently selected order
OrderSymbol	Returns symbol name of the currently selected order
OrderTakeProfit	Returns take profit value of the currently selected order
OrderTicket	Returns ticket number of the currently selected order
OrderType	Returns order operation type of the currently selected order

OrderClose

Closes opened order.

```
bool  OrderClose(   int          ticket,      // ticket
   double      lots,        // volume
   double      price,       // close price
   int         slippage,    // slippage
   color       arrow_color  // color
   );
```

Parameters

ticket

[in] Unique number of the order ticket.

lots

[in] Number of lots.

price

[in] Closing price.

slippage

[in] Value of the maximum price slippage in points.

arrow_color

[in] Color of the closing arrow on the chart. If the parameter is missing or has CLR_NONE value closing arrow will not be drawn on the chart.

Returned value

Returns true if successful, otherwise false. To get additional error information, one has to call the GetLastError() function.

Example:

```
  if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
    {
     OrderClose(order_id,1,Ask,3,Red);
     return(0);
    }
```

OrderCloseBy

Closes an opened order by another opposite opened order.

```
bool  OrderCloseBy(    int          ticket,       // ticket to close
   int         opposite,    // opposite ticket
   color       arrow_color  // color
   );
```

Parameters

ticket

[in] Unique number of the order ticket.

opposite

[in] Unique number of the opposite order ticket.

arrow_color

[in] Color of the closing arrow on the chart. If the parameter is missing or has CLR_NONE value closing arrow will not be drawn on the chart.

Returned value

Returns true if successful, otherwise false. To get additional error information, one has to call the GetLastError() function.

Example:

```
  if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
    {
     OrderCloseBy(order_id,opposite_id);
     return(0);
    }
```

OrderClosePrice

Returns close price of the currently selected order.

```
double  OrderClosePrice();
```

Returned value

The close price of currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  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());
```

OrderCloseTime

Returns close time of the currently selected order.

```
datetime  OrderCloseTime();
```

Returned value

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. Open and pending orders close time is equal to 0.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  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());
```

OrderComment

Returns comment of the currently selected order.

```
string  OrderComment();
```

Returned value

Comment of the currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  string comment;   if(OrderSelect(10,SELECT_BY_TICKET)==false)
    {
     Print("OrderSelect failed error code is",GetLastError());
     return(0);
    }
  comment = OrderComment();
  // ...
```

OrderCommission

Returns calculated commission of the currently selected order.

```
double  OrderCommission();
```

Returned value

The calculated commission of the currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(10,SELECT_BY_POS)==true)     Print("Commission for the or
  else
    Print("OrderSelect failed error code is",GetLastError());
```

OrderDelete

Deletes previously opened pending order.

```
bool  OrderDelete(    int          ticket,       // ticket
   color       arrow_color   // color
   );
```

Parameters

ticket

[in] Unique number of the order ticket.

arrow_color

[in] Color of the arrow on the chart. If the parameter is missing or has CLR_NONE value arrow will not be drawn on the chart.

Returned value

If the function succeeds, it returns true, otherwise false. To get the detailed error information, call the GetLastError() function.

Example:

```
  if(Ask>var1)
    {
     OrderDelete(order_ticket);
     return(0);
    }
```

OrderExpiration

Returns expiration date of the selected pending order.

```
datetime  OrderExpiration();
```

Returned value

Expiration date of the selected pending order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(10, SELECT_BY_TICKET)==true)     Print("Order expiration :
  else
    Print("OrderSelect returned error of ",GetLastError());
```

OrderLots

Returns amount of lots of the selected order.

```
double  OrderLots();
```

Returned value

Amount of lots (trade volume) of the selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(10,SELECT_BY_POS)==true)     Print("lots for the order 10
  else
    Print("OrderSelect returned error of ",GetLastError());
```

OrderMagicNumber

Returns an identifying (magic) number of the currently selected order.

```
int  OrderMagicNumber();
```

Returned value

The identifying (magic) number of the currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(10,SELECT_BY_POS)==true)     Print("Magic number for the
  else
    Print("OrderSelect returned error of ",GetLastError());
```

OrderModify

Modification of characteristics of the previously opened or pending orders.

```
bool  OrderModify(    int          ticket,       // ticket
   double      price,        // price
   double      stoploss,     // stop loss
   double      takeprofit,   // take profit
   datetime    expiration,   // expiration
   color       arrow_color   // color
   );
```

Parameters

ticket

[in] Unique number of the order ticket.

price

[in] New open price of the pending order.

stoploss

[in] New StopLoss level.

takeprofit

[in] New TakeProfit level.

expiration

[in] Pending order expiration time.

arrow_color

[in] Arrow color for StopLoss/TakeProfit modifications in the chart. If the parameter is missing or has CLR_NONE value, the arrows will not be shown in the chart.

Returned value

If the function succeeds, it returns true, otherwise false. To get the detailed error information, call the GetLastError() function.

Note

Open price and expiration time can be changed only for pending orders. If unchanged values are passed as the function parameters, the error 1 (ERR_NO_RESULT) will be generated.

Pending order expiration time can be disabled in some trade servers. In this case, when a non-zero value is specified in the expiration parameter, the error 147 (ERR_TRADE_EXPIRATION_DENIED) will be generated.

Example:

```
void OnStart()
  {
   int TrailingStop=50;
//--- modifies Stop Loss price for buy order №12345
   if(TrailingStop>0)
     {
      OrderSelect(12345,SELECT_BY_TICKET);
      if(Bid-OrderOpenPrice()>Point*TrailingStop)
        {
         if(OrderStopLoss()<Bid-Point*TrailingStop)
           {
            bool res=OrderModify(OrderTicket(),OrderOpenPrice(),NormalizeD
            if(!res)
               Print("Error in OrderModify. Error code=",GetLastError());
            else
               Print("Order modified successfully.");
           }
        }
     }
  }
```

OrderOpenPrice

Returns open price of the currently selected order.

```
double  OrderOpenPrice();
```

Returned value

Open price of the currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(10, SELECT_BY_POS)==true)     Print("open price for the o
  else
    Print("OrderSelect returned the error of ",GetLastError());
```

OrderOpenTime

Returns open time of the currently selected order.

```
datetime  OrderOpenTime();
```

Returned value

Open time of the currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(10, SELECT_BY_POS)==true)     Print("open time for the ord
  else
    Print("OrderSelect returned error of ",GetLastError());
```

OrderPrint

Prints information about the selected order in the log.

```
void  OrderPrint();
```

Parameters

Prints information about the selected order in the log in the following format:

#ticket number; open time; trade operation; amount of lots; symbol; open price; Stop Loss; Take Profit; close time; close price; commission; swap; profit; comment; magic number; pending order expiration date.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(10, SELECT_BY_TICKET)==true)     OrderPrint();
  else
    Print("OrderSelect failed error code is",GetLastError());
```

OrderProfit

Returns profit of the currently selected order.

```
double  OrderProfit();
```

Returned value

The net profit value (without swaps or commissions) for the selected order. For open orders, it is the current unrealized profit. For closed orders, it is the fixed profit.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(10, SELECT_BY_POS)==true)     Print("Profit for the order
  else
    Print("OrderSelect returned the error of ",GetLastError());
```

OrderSelect

The function selects an order for further processing.

```
bool  OrderSelect(   int      index,            // index or order ticket
   int      select,           // flag
   int      pool=MODE_TRADES  // mode
   );
```

Parameters

ticket

[in] Order index or order ticket depending on the second parameter.

select

[in] 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=MODE_TRADES

[in] Optional order pool index. Used when the selected 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).

Returned value

It returns true if the function succeeds, otherwise falses. To get the error information, one has to call the GetLastError() function.

Note

The pool parameter is ignored if the order is selected by the ticket number. The ticket number is a unique order identifier.

To find out from what list the order has been selected, its close time must be analyzed. If the order close time equals to 0, the order is open or pending and taken from the terminal open orders list.

One can distinguish an opened order from a pending order by the order type. If the order close time does not equal to 0, the order is a closed order or a deleted pending order and was selected from the terminal history. They also differ from each other by their order types.

The OrderSelect() function copies order data into program environment and all further calls of OrderClosePrice(), OrderCloseTime(), OrderComment(), OrderCommission(), OrderExpiration(), OrderLots(), OrderMagicNumber(), OrderOpenPrice(), OrderOpenTime(), OrderPrint(), OrderProfit(), OrderStopLoss(), OrderSwap(), OrderSymbol(), OrderTakeProfit(), OrderTicket(), OrderType() functions return the data, copied earlier. It means that in some cases the order details (open price, SL/TP levels or expiration date) may change and the data become non-actual. It is strongly recommended to call the OrderSelect() function before request the order data.

Consecutive selection of orders using the SELECT_BY_POS parameter returns information in the sequence in which it was received from the trading server. Sorting of the resulting list of orders cannot be guaranteed.

Example:

```
if(OrderSelect(12470, SELECT_BY_TICKET)==true)
  {
   Print("order #12470 open price is ", OrderOpenPrice());
   Print("order #12470 close price is ", OrderClosePrice());
  }
else
  Print("OrderSelect returned the error of ",GetLastError());
```

See also

Order properties, OrderClosePrice(), OrderCloseTime(), OrderComment(), OrderCommission(), OrderExpiration(), OrderLots(), OrderMagicNumber(), OrderOpenPrice(), OrderOpenTime(), OrderPrint(), OrderProfit(), OrderStopLoss(), OrderSwap(), OrderSymbol(), OrderTakeProfit(), OrderTicket(), OrderType()

OrderSend

The main function used to open market or place a pending order.

```
int  OrderSend(   string   symbol,              // symbol
   int      cmd,                 // operation
   double   volume,              // volume
   double   price,               // price
   int      slippage,            // slippage
   double   stoploss,            // stop loss
   double   takeprofit,          // take profit
   string   comment=NULL,        // comment
   int      magic=0,             // magic number
   datetime expiration=0,        // pending order expiration
   color    arrow_color=clrNONE  // color
   );
```

Parameters

symbol

[in] Symbol for trading.

cmd

[in] Operation type. It can be any of the Trade operation enumeration.

volume

[in] Number of lots.

price

[in] Order price.

slippage

[in] Maximum price slippage for buy or sell orders.

stoploss

[in] Stop loss level.

takeprofit

[in] Take profit level.

comment=NULL

[in] Order comment text. Last part of the comment may be changed by server.

magic=0

[in] Order magic number. May be used as user defined identifier.

expiration=0

[in] Order expiration time (for pending orders only).

arrow_color=clrNONE

[in] Color of the opening arrow on the chart. If parameter is missing or has CLR_NONE value opening arrow is not drawn on the chart.

Returned value

Returns number of the ticket assigned to the order by the trade server or -1 if it fails. To get additional error information, one has to call the GetLastError() function.

Note

At opening of a market order (OP_SELL or OP_BUY), only the latest prices of Bid (for selling) or Ask (for buying) can be used as open price. If operation is performed with a security differing from the current one, the MarketInfo() function must be used with MODE_BID or MODE_ASK parameter for the latest quotes for this security to be obtained.

Calculated or unnormalized price cannot be applied. If there has not been the requested open price in the price thread or it has not been normalized according to the amount of digits after decimal point, the error 129 (ERR_INVALID_PRICE) will be generated. If the requested open price is fully out of date, the error 138 (ERR_REQUOTE) will be generated independently on the slippage parameter. If the requested price is out of date, but present in the thread, the order will be opened at the current price and only if the current price lies within the range of price+-slippage.

StopLoss and TakeProfit levels cannot be too close to the market. The minimal distance of stop levels in points can be obtained using the MarketInfo() function with MODE_STOPLEVEL parameter. In the case of erroneous or unnormalized stop levels, the error 130 (ERR_INVALID_STOPS) will be generated. A zero value of MODE_STOPLEVEL means either absence of any restrictions on the minimal distance for Stop Loss/Take Profit or the fact that a trade server utilizes some external mechanisms for dynamic level control, which cannot be translated in the client terminal. In the second case, GetLastError() can return error 130, because MODE_STOPLEVEL is actually "floating" here.

At placing of a pending order, the open price cannot be too close to the market. The minimal distance of the pending price from the current market one in points can be obtained using the MarketInfo() function with the MODE_STOPLEVEL parameter. In case of false open price of a pending order, the error 130 (ERR_INVALID_STOPS) will be generated.

Applying of pending order expiration time can be disabled in some trade servers. In this case, when a non-zero value is specified in the expiration parameter, the error 147 (ERR_TRADE_EXPIRATION_DENIED) will be generated.

On some trade servers, the total amount of open and pending orders can be limited. If this limit has been exceeded, no new order will be opened (or no pending order will be placed) and trade server will return error 148 (ERR_TRADE_TOO_MANY_ORDERS).

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- get minimum stop level
   double minstoplevel=MarketInfo(Symbol(),MODE_STOPLEVEL);
   Print("Minimum Stop Level=",minstoplevel," points");
   double price=Ask;
//--- calculated SL and TP prices must be normalized
   double stoploss=NormalizeDouble(Bid-minstoplevel*Point,Digits);
   double takeprofit=NormalizeDouble(Bid+minstoplevel*Point,Digits);
//--- place market order to buy 1 lot
   int ticket=OrderSend(Symbol(),OP_BUY,1,price,3,stoploss,takeprofit,"My
   if(ticket<0)
     {
      Print("OrderSend failed with error #",GetLastError());
     }
   else
      Print("OrderSend placed successfully");
//---
  }
```

OrdersHistoryTotal

Returns the number of closed orders in the account history loaded into the terminal.

```
int  OrdersHistoryTotal();
```

Returned value

The number of closed orders in the account history loaded into the terminal. The history list size depends on the current settings of the "Account history" tab of the terminal.

Example:

```
  // retrieving info from trade history   int i,hstTotal=OrdersHistoryTota
  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
    }
```

OrderStopLoss

Returns stop loss value of the currently selected order.

```
double  OrderStopLoss();
```

Returned value

Stop loss value of the currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(ticket,SELECT_BY_POS)==true)     Print("Stop loss value fo
  else
    Print("OrderSelect failed error code is",GetLastError());
```

OrdersTotal

Returns the number of market and pending orders.

```
int  OrdersTotal();
```

Returned value

Total amount of market and pending orders.

Example:

```
   int handle=FileOpen("OrdersReport.csv",FILE_WRITE|FILE_CSV,"\t");   if(ha
   // 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(),Order
     }
   FileClose(handle);
```

OrderSwap

Returns swap value of the currently selected order.

```
double  OrderSwap();
```

Returned value

Swap value of the currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(order_id, SELECT_BY_TICKET)==true)     Print("Swap for the
  else
    Print("OrderSelect failed error code is",GetLastError());
```

OrderSymbol

Returns symbol name of the currently selected order.

```
string  OrderSymbol();
```

Returned value

The symbol name of the currently selected order.

Note

The order must be previously selected by the [OrderSelect()]() function.

Example:

```
  if(OrderSelect(12, SELECT_BY_POS)==true)     Print("symbol of order #", 
  else
    Print("OrderSelect failed error code is",GetLastError());
```

OrderTakeProfit

Returns take profit value of the currently selected order.

```
double  OrderTakeProfit();
```

Returned value

Take profit value of the currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(12, SELECT_BY_POS)==true)     Print("Order #",OrderTicket
  else
    Print("OrderSelect() returns error - ",GetLastError());
```

OrderTicket

Returns ticket number of the currently selected order.

```
int  OrderTicket();
```

Returned value

Ticket number of the currently selected order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
  if(OrderSelect(12, SELECT_BY_POS)==true)     order=OrderTicket();
  else
    Print("OrderSelect failed error code is",GetLastError());
```

OrderType

Returns order operation type of the currently selected order.

```
int  OrderType();
```

Returned value

Order operation type of the currently selected order. It can be any of the following values:

OP_BUY - buy order, OP_SELL - sell order,
OP_BUYLIMIT - buy limit pending order,
OP_BUYSTOP - buy stop pending order,
OP_SELLLIMIT - sell limit pending order,
OP_SELLSTOP - sell stop pending order.

Note

The order must be previously selected by the OrderSelect() function.

Example:

```
int order_type;
if(OrderSelect(12, SELECT_BY_POS)==true)
  {
   order_type=OrderType();
   // ...
  }
else
  Print("OrderSelect() returned error - ",GetLastError());
```

Trade Signals

This is the group of functions intended for managing trade signals. The functions allow:

- get information about trade signals, available for copying,
- get and set the signal copy settings,
- subscribe and unsubscribe to the signal copying using MQL4 language functions.

Function	Action
SignalBaseGetDouble	Returns the value of double type property for selected signal
SignalBaseGetInteger	Returns the value of integer type property for selected signal
SignalBaseGetString	Returns the value of string type property for selected signal
SignalBaseSelect	Selects a signal from signals, available in terminal for further working with it
SignalBaseTotal	Returns the total amount of signals, available in terminal
SignalInfoGetDouble	Returns the value of double type property of signal copy settings
SignalInfoGetInteger	Returns the value of integer type property of signal copy settings
SignalInfoGetString	Returns the value of string type property of signal copy settings
SignalInfoSetDouble	Sets the value of double type property of signal copy settings
SignalInfoSetInteger	Sets the value of integer type property of signal copy settings
SignalSubscribe	Subscribes to the trading signal
SignalUnsubscribe	Cancels subscription

SignalBaseGetDouble

Returns the value of double type property for selected signal.

```
double  SignalBaseGetDouble(    ENUM_SIGNAL_BASE_DOUBLE      property_id
   );
```

Parameters

property_id

[in] Signal property identifier. The value can be one of the values of the ENUM_SIGNAL_BASE_DOUBLE enumeration.

Returned value

The value of double type property of the selected signal.

SignalBaseGetInteger

Returns the value of integer type property for selected signal.

```
long  SignalBaseGetInteger(    ENUM_SIGNAL_BASE_INTEGER     property_id
   );
```

Parameters

property_id

[in] Signal property identifier. The value can be one of the values of the ENUM_SIGNAL_BASE_INTEGER enumeration.

Returned value

The value of integer type property of the selected signal.

SignalBaseGetString

Returns the value of string type property for selected signal.

```
string  SignalBaseGetString(    ENUM_SIGNAL_BASE_STRING     property_id
   );
```

Parameters

property_id

[in] Signal property identifier. The value can be one of the values of the ENUM_SIGNAL_BASE_STRING enumeration.

Returned value

The value of string type property of the selected signal.

SignalBaseSelect

Selects a signal from signals, available in terminal for further working with it.

```
bool  SignalBaseSelect(    int      index      // signal index
   );
```

Parameters

index

[in] Signal index in base of trading signals.

Returned value

Returns true if successful, otherwise returns false. To read more about the error call GetLastError().

Example:

```
void OnStart()
  {
//--- get total amount of signals in the terminal
   int total=SignalBaseTotal();
//--- process all signals
   for(int i=0;i<total;i++)
     {
      //--- select the signal by index
      if(SignalBaseSelect(i))
        {
         //--- get signal properties
         long   id    =SignalBaseGetInteger(SIGNAL_BASE_ID);          // s
         long   pips  =SignalBaseGetInteger(SIGNAL_BASE_PIPS);        // p
         long   subscr=SignalBaseGetInteger(SIGNAL_BASE_SUBSCRIBERS); // n
         string name  =SignalBaseGetString(SIGNAL_BASE_NAME);         // s
         double price =SignalBaseGetDouble(SIGNAL_BASE_PRICE);        // s
         string curr  =SignalBaseGetString(SIGNAL_BASE_CURRENCY);     // s
         //--- print all profitable free signals with subscribers
         if(price==0.0 && pips>0 && subscr>0)
            PrintFormat("id=%d, name=\"%s\", currency=%s, pips=%d, subscri
        }
      else PrintFormat("Error in call of SignalBaseSelect. Error code=%d",
     }
  }
```

SignalBaseTotal

Returns the total amount of signals, available in terminal.

```
int  SignalBaseTotal();
```

Returned value

The total amount of signals, available in terminal.

SignalInfoGetDouble

Returns the value of double type property of signal copy settings.

```
double  SignalInfoGetDouble(    ENUM_SIGNAL_INFO_DOUBLE     property_id
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_DOUBLE enumeration.

Returned value

The value of double type property of signal copy settings.

SignalInfoGetInteger

Returns the value of integer type property of signal copy settings.

```
long  SignalInfoGetInteger(    ENUM_SIGNAL_INFO_INTEGER     property_id
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_INTEGER enumeration.

Returned value

The value of integer type property of signal copy settings.

SignalInfoGetString

Returns the value of string type property of signal copy settings.

```
string  SignalInfoGetString(    ENUM_SIGNAL_INFO_STRING      property_id
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_STRING enumeration.

Returned value

The value of string type property of signal copy settings.

SignalInfoSetDouble

Sets the value of double type property of signal copy settings.

```
bool  SignalInfoSetDouble(    ENUM_SIGNAL_INFO_DOUBLE       property_id,
   double                             value             // new value
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_DOUBLE enumeration.

value

[in] The value of signal copy settings property.

Returned value

Returns true if property has been changed, otherwise returns false. To read more about the error call GetLastError().

SignalInfoSetInteger

Sets the value of integer type property of signal copy settings.

```
bool  SignalInfoSetInteger(    ENUM_SIGNAL_INFO_INTEGER     property_id,
   long                                value             // new value
   );
```

Parameters

property_id

[in] Signal copy settings property identifier. The value can be one of the values of the ENUM_SIGNAL_INFO_INTEGER enumeration.

value

[in] The value of signal copy settings property.

Returned value

Returns true if property has been changed, otherwise returns false. To read more about the error call GetLastError().

SignalSubscribe

Subscribes to the trading signal.

```
bool   SignalSubscribe(    long      signal_id      // signal id
   );
```

Parameters

signal_id

[in] Signal identifier.

Returned value

Returns true if subscription was successful, otherwise returns false. To read more about the error call GetLastError().

SignalUnsubscribe

Cancels subscription.

```
bool  SignalUnsubscribe();
```

Returned value

Returns true if subscription has been canceled successfully, otherwise returns false. To read more about the error call GetLastError().

Global Variables of the Client Terminal

There is a group set of functions for working with global variables.

Global variables of the client terminal should not be mixed up with variables declared in the global scope of the mql4 program.

Global variables are kept in the client terminal for 4 weeks since the last access, then they will be deleted automatically. An access to a global variable is not only setting of a new value, but reading of the global variable value, as well.

Global variables of the client terminal are accessible simultaneously from all mql4 programs launched in the client terminal.

When testing and optimizing the Expert Advisors that use global variables, keep in mind that client terminal and the Strategy Tester share common global variables. Therefore, the names of the global variables must be different from the names of the global variables used by other mql4 programs. Otherwise, it may lead to incorrect work of mql4 programs and inaccurate testing results.

Function	Action
GlobalVariableCheck	Checks the existence of a global variable with the specified name
GlobalVariableTime	Returns time of the last accessing the global variable
GlobalVariableDel	Deletes a global variable
GlobalVariableGet	Returns the value of a global variable
GlobalVariableName	Returns the name of a global variable by its ordinal number in the list of global variables
GlobalVariableSet	Sets the new value to a global variable
GlobalVariablesFlush	Forcibly saves contents of all global variables to a disk
GlobalVariableTemp	Sets the new value to a global variable, that exists only in the current session of the terminal
GlobalVariableSetOnCondition	Sets the new value of the existing global variable by condition
GlobalVariablesDeleteAll	Deletes global variables with the specified prefix in their names
GlobalVariablesTotal	Returns the total number of global variables

GlobalVariableCheck

Checks the existence of a global variable with the specified name

```
bool  GlobalVariableCheck(    string  name       // Global variable name
   );
```

Parameters

name

[in] Global variable name.

Return Value

Returns true, if the global variable exists, otherwise returns false.

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

See also

GlobalVariableTime()

GlobalVariableTime

Returns the time when the global variable was last accessed.

```
datetime  GlobalVariableTime(    string  name      // name
   );
```

Parameters

name

[in] Name of the global variable.

Return Value

The function returns time of last accessing the specified global variable. Addressing a variable for its value, for example using the GlobalVariableGet() and GlobalVariableCheck() functions, also modifies the time of last access. In order to obtain error details, call the GetLastError() function.

Note

Global variables exist in the client terminal during 4 weeks since they were called last. After that they are automatically deleted.

See also

GlobalVariableCheck()

GlobalVariableDel

Deletes a global variable from the client terminal.

```
bool  GlobalVariableDel(    string  name       // Global variable name
   );
```

Parameters

name

[in] Global variable name.

Return Value

If successful, the function returns true, otherwise it returns false. To obtain an information about the error it is necessary to call the function GetLastError().

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariableGet

Returns the value of an existing global variable of the client terminal. There are 2 variants of the function.

1. Immediately returns the value of the global variable.

```
double  GlobalVariableGet(    string  name       // Global variable name
   );
```

2. Returns true or false depending on the success of the function run. If successful, the global variable of the client terminal is placed in a variable passed by reference in the second parameter.

```
bool  GlobalVariableGet(
   string  name,                 // Global variable name
   double& double_var            // This variable will contain the value of t
   );
```

Parameters

name

[in] Global variable name.

double_var

[out] Target variable of the double type, which accepts the value stored in a the global variable of the client terminal.

Return Value

The value of the existing global variable or 0 in case of an error. For more details about the error, call GetLastError().

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariableName

Returns the name of a global variable by its ordinal number.

```
string  GlobalVariableName(    int   index      // Global variable number i
   );
```

Parameters

index

[in] Sequence number in the list of global variables. It should be greater than or equal to 0 and less than GlobalVariablesTotal().

Return Value

Global variable name by its ordinal number in the list of global variables. For more details about the error, call GetLastError().

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariableSet

Sets a new value for a global variable. If the variable does not exist, the system creates a new global variable.

```
datetime  GlobalVariableSet(    string  name,       // Global variable name
   double  value       // Value to set
   );
```

Parameters

name

[in] Global variable name.

value

[in] The new numerical value.

Return Value

If successful, the function returns the last modification time, otherwise 0. For more details about the error, call GetLastError().

Note

A global variable name should not exceed 63 characters. Characters not belonging to the current code page are not allowed (characters that cannot be converted from Unicode to ANSI are replaced with '?'). If programs are to be distributed among users with different code pages, we strongly recommend using Latin characters in global variable names.

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariablesFlush

Forcibly saves contents of all global variables to a disk.

```
void  GlobalVariablesFlush();
```

Return Value

No return value.

Note

The terminal writes all the global variables when the work is over, but data can be lost at a sudden computer operation failure. This function allows independently controlling the process of saving global variables in case of contingency.

GlobalVariableTemp

The function attempts to create a temporary global variable. If the variable doesn't exist, the system creates a new temporary global variable.

```
bool  GlobalVariableTemp(    string  name       // Global variable name
   );
```

Parameters

name

[in] The name of a temporary global variable.

Return Value

If successful, the function returns true, otherwise - false. To get details about the error, you should call the GetLastError() function.

Note

Temporary global variables exist only while the client terminal is running; after the terminal shutdown they are automatically deleted. Note that during the execution of GlobalVariablesFlush() temporary global variables are not written to a disk.

After a temporary global variable has been created, it can be accessed and modified the same as global variable of the client terminal.

GlobalVariableSetOnCondition

Sets the new value of the existing global variable if the current value equals to the third parameter check_value. If there is no global variable, the function will generate an error ERR_GLOBALVARIABLE_NOT_FOUND (4501) and return false.

```
bool  GlobalVariableSetOnCondition(    string  name,            // Global
   double  value,            // New value for variable if condition is true
   double  check_value       // Check value condition
   );
```

Parameters

name

[in] The name of a global variable.

value

[in] New value.

check_value

[in] The value to check the current value of the global variable.

Return Value

If successful, the function returns true, otherwise it returns false. For details about the error call GetLastError(). If the current value of the global variable is different from check_value, the function returns false.

Note

Function provides atomic access to the global variable, so it can be used for providing of a mutex at interaction of several Expert Advisors working simultaneously within one client terminal.

GlobalVariablesDeleteAll

Deletes global variables of the client terminal.

```
int  GlobalVariablesDeleteAll(   string     prefix_name=NULL,     // All c
   datetime    limit_data=0            // All global variables that were char
   );
```

Parameters

prefix_name=NULL

[in] Name prefix global variables to remove. If you specify a prefix NULL or empty string, then all variables that meet the data criterion will be deleted.

limit_data=0

[in] Optional parameter. Date to select global variables by the time of their last modification. The function removes global variables, which were changed before this date. If the parameter is zero, then all variables that meet the first criterion (prefix) are deleted.

Return Value

The number of deleted variables.

Note

If both options are equal to zero (prefix_name = NULL and limit_data = 0), then function deletes all global variables of the terminal. If both parameters are specified, then it deletes global variables corresponding to both parameters.

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted.

GlobalVariablesTotal

Returns the total number of global variables of the client terminal.

```
int  GlobalVariablesTotal();
```

Return Value

Number of global variables.

Note

Global variables exist in the client terminal during 4 weeks since their last use, then they are automatically deleted. Call of a global variable is not only setting a new value, but also reading the value of the global variable.

File Functions

This is a group of functions for working with files.

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means cannot be outside the file sandbox.

There are two directories (with subdirectories) in which working files can be located:

- terminal_data_folder\MQL4\Files\ (in the terminal menu select to view "File" - "Open the data directory");
- the common folder for all the terminals installed on a computer - usually located in the directory C:\Documents and Settings\All Users\Application Data\MetaQuotes\Terminal\Common\Files.

There is a program method to obtain names of these catalogs using the TerminalInfoString() function, using the ENUM_TERMINAL_INFO_STRING enumeration:

```
//--- Folder that stores the terminal data   string terminal_data_path=Te
//--- Common folder for all client terminals
   string common_data_path=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
```

Work with files from other directories is prohibited.

File functions allow working with so-called "named pipes". To do this, simply call FileOpen() function with appropriate parameters.

Function	Action
FileFindFirst	Starts the search of files in a directory in accordance with the specified filter
FileFindNext	Continues the search started by the FileFindFirst() function
FileFindClose	Closes search handle
FileOpen	Opens a file with a specified name and flag
FileDelete	Deletes a specified file
FileFlush	Writes to a disk all data remaining in the input/output file buffer
FileGetInteger	Gets an integer property of a file
FileIsEnding	Defines the end of a file in the process of reading
FileIsLineEnding	Defines the end of a line in a text file in the process of reading

FileClose	Closes a previously opened file
FileIsExist	Checks the existence of a file
FileCopy	Copies the original file from a local or shared folder to another file
FileMove	Moves or renames a file
FileReadArray	Reads arrays of any type except for string from the file of the BIN type
FileReadBool	Reads from the file of the CSV type a string from the current position till a delimiter (or till the end of a text line) and converts the read string to a value of bool type
FileReadDatetime	Reads from the file of the CSV type a string of one of the formats: "YYYY.MM.DD HH:MM:SS", "YYYY.MM.DD" or "HH:MM:SS" - and converts it into a datetime value
FileReadDouble	Reads a double value from the current position of the file pointer
FileReadFloat	Reads a float value from the current position of the file pointer
FileReadInteger	Reads int, short or char value from the current position of the file pointer
FileReadLong	Reads a long type value from the current position of the file pointer
FileReadNumber	Reads from the file of the CSV type a string from the current position till a delimiter (or til the end of a text line) and converts the read string into double value
FileReadString	Reads a string from the current position of a file pointer from a file
FileReadStruct	Reads the contents from a binary file into a structure passed as a parameter, from the current position of the file pointer
FileSeek	Moves the position of the file pointer by a specified number of bytes relative to the specified position
FileSize	Returns the size of a corresponding open file
FileTell	Returns the current position of the file pointer of a corresponding open file
FileWrite	Writes data to a file of CSV or TXT type
FileWriteArray	Writes arrays of any type except for string into a file of BIN type
FileWriteDouble	Writes value of the double type from the current position of a file pointer into a binary file
FileWriteFloat	Writes value of the float type from the current position of a file pointer into a binary file

FileWriteInteger	Writes value of the int type from the current position of a file pointer into a binary file
FileWriteLong	Writes value of the long type from the current position of a file pointer into a binary file
FileWriteString	Writes the value of a string parameter into a BIN or TXT file starting from the current position of the file pointer
FileWriteStruct	Writes the contents of a structure passed as a parameter into a binary file, starting from the current position of the file pointer
FolderCreate	Creates a folder in the Files directory
FolderDelete	Removes a selected directory. If the folder is not empty, then it can't be removed
FolderClean	Deletes all files in the specified folder
FileOpenHistory	Opens file in the current history directory or in its subfolders

If the file is opened for writing using FileOpen(), all subfolders specified in the path will be created if there are no such ones.

FileFindFirst

The function starts the search of files or subdirectories in a directory in accordance with the specified filter.

```
long  FileFindFirst(    const string   file_filter,           // String - se
   string&         returned_filename,    // Name of the file or subdirector
   int             common_flag=0         // Defines the search
   );
```

Parameters

file_filter

[in] Search filter. A subdirectory (or sequence of nested subdirectories) relative to the \Files directory, in which files should be searched for, can be specified in the filter.

returned_filename

[out] The returned parameter, where, in case of success, the name of the first found file or subdirectory is placed. Only the file name is returned (including the extension), the directories and subdirectories are not included no matter if they are specified or not in the search filter.

common_flag

[in] Flag determining the location of the file. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder.

Return Value

Returns handle of the object searched, which should be used for further sorting of files and subdirectories by the FileFindNext() function, or INVALID_HANDLE when there is no file and subdirectory corresponding to the filter (in the particular case - when the directory is empty). After searching, the handle must be closed using the FileFindClose() function.

Note

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means, cannot be outside the file sandbox.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- filter
input string InpFilter="Dir1\\*";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   string int_dir="";
   int    i=1,pos=0,last_pos=-1;
//--- search for the last backslash
   while(!IsStopped())
     {
      pos=StringFind(InpFilter,"\\",pos+1);
      if(pos>=0)
         last_pos=pos;
      else
         break;
     }
//--- the filter contains the folder name
   if(last_pos>=0)
      int_dir=StringSubstr(InpFilter,0,last_pos+1);
//--- get the search handle in the root of the local folder
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- check if the FileFindFirst() is executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- in a loop, check if the passed strings are the names of files
      do
        {
         ResetLastError();
         //--- if it's a file, the function returns true, and if it's a di
         FileIsExist(int_dir+file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==ERR_FILE_IS_DIR
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- close the search handle
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

See also

FileFindNext(), FileFindClose()

FileFindNext

The function continues the search started by FileFindFirst().

```
bool  FileFindNext(    long      search_handle,          // Search handle
   string&    returned_filename      // Name of the file or subdirectory fo
   );
```

Parameters

search_handle

[in] Search handle, retrieved by FileFindFirst().

returned_filename

[out] The name of the next file or subdirectory found. Only the file name is returned (including the extension), the directories and subdirectories are not included no matter if they are specified or not in the search filter.

Return Value

If successful returns true, otherwise false.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- filter
input string InpFilter="*";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   int    i=1;
//--- receive search handle in local folder's root
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- check if FileFindFirst() function executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- check if the passed strings are file or directory names in the
      do
        {
         ResetLastError();
         //--- if this is a file, the function will return true, if it is
         FileIsExist(file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==ERR_FILE_IS_DIR
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- close search handle
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

See also

[FileFindFirst()](), [FileFindClose()]()

FileFindClose

The function closes the search handle.

```
void  FileFindClose(    long   search_handle       //   Search handle
   );
```

Parameters

search_handle

[in] Search handle, retrieved by FileFindFirst().

Return Value

No value returned.

Note

Function must be called to free up system resources.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- filter
input string InpFilter="*";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   int    i=1;
//--- receive search handle in local folder's root
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- check if FileFindFirst() function executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- check if the passed strings are file or directory names in the
      do
        {
         ResetLastError();
         //--- if this is a file, the function will return true, if it is
         FileIsExist(file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==5018 ? "Directo
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- close search handle
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

See also

FileFindFirst(), FileFindNext()

FileIsExist

Checks the existence of a file.

```
bool  FileIsExist(    const string  file_name,         // File name
   int             common_flag=0    // Search area
   );
```

Parameters

file_name

[in] The name of the file being checked

common_flag=0

[in] Flag determining the location of the file. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder.

Return Value

Returns true, if the specified file exists.

Note

Checked file can turn out to be a subdirectory. In this case, FileIsExist() function will return false, while error 5019 will be logged in _LastError variable - "This is a directory, not a file" (see example for FileFindFirst() function).

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means, cannot be outside the file sandbox.

If common_flag = FILE_COMMON, then the function looks for the file in a shared folder for all client terminals \Terminal\Common\Files, otherwise the function looks for a file in a local folder (MQL4\Files or MQL4\Tester\Files in the case of testing).

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- date for old files
input datetime InpFilesDate=D'2013.01.01 00:00';
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // variable for storing file names
   string   filter="*.txt"; // filter for searching the files
   datetime create_date;    // file creation date
   string   files[];        // list of file names
   int      def_size=25;    // array size by default
   int      size=0;         // number of files
//--- allocate memory for the array
   ArrayResize(files,def_size);
//--- receive the search handle in the local folder's root
   long search_handle=FileFindFirst(filter,file_name);
//--- check if FileFindFirst() executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- searching files in the loop
      do
        {
         files[size]=file_name;
         //--- increase the array size
         size++;
         if(size==def_size)
           {
            def_size+=25;
            ArrayResize(files,def_size);
           }
         //--- reset the error value
         ResetLastError();
         //--- receive the file creation date
         create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,f
         //--- check if the file is old
         if(create_date<InpFilesDate)
           {
            PrintFormat("%s file deleted!",file_name);
            //--- delete the old file
            FileDelete(file_name);
           }
        }
      while(FileFindNext(search_handle,file_name));
      //--- close the search handle
      FileFindClose(search_handle);
     }
   else
     {
      Print("Files not found!");
      return;
     }
//--- check what files have remained
   PrintFormat("Results:");
   for(int i=0;i<size;i++)
     {
      if(FileIsExist(files[i]))
         PrintFormat("%s file exists!",files[i]);
      else
         PrintFormat("%s file deleted!",files[i]);
     }
  }
```

See also

FileFindFirst()

FileOpen

The function opens the file with the specified name and flag.

```
int  FileOpen(    string  file_name,          // File name
   int      open_flags,          // Combination of flags
   short    delimiter=';',       // Delimiter
   uint     codepage=CP_ACP      // Code page
   );
```

Parameters

file_name

[in] The name of the file can contain subfolders. If the file is opened for writing, these subfolders will be created if there are no such ones.

open_flags

[in] combination of flags determining the operation mode for the file. The flags are defined as follows:
FILE_READ file is opened for reading
FILE_WRITE file is opened for writing
FILE_BIN binary read-write mode (no conversion from a string and to a string)
FILE_CSV file of csv type (all recorded items are converted to the strings of unicode or ansi type, and are separated by a delimiter)
FILE_TXT a simple text file (the same as csv, but the delimiter is not taken into account)
FILE_ANSI lines of ANSI type (single-byte symbols)
FILE_UNICODE lines of UNICODE type (double-byte characters)
FILE_SHARE_READ shared reading from several programs
FILE_SHARE_WRITE shared writing from several programs
FILE_COMMON location of the file in a shared folder for all client terminals \Terminal\Common\Files

delimiter=';'

[in] value to be used as a separator in txt or csv-file. If the csv-file delimiter is not specified, the default delimiter is ";". If the txt-file delimiter is not specified, then no separator is used. If the separator is clearly set to 0, then no separator is used.

codepage=CP_ACP

[in] Optional parameter. The value of the code page. For the most-used code pages provide appropriate constants.

Return Value

If a file has been opened successfully, the function returns the file handle, which is then used to access the file data. In case of failure returns INVALID_HANDLE.

Note

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means, cannot be outside the file sandbox.

The file is opened in the folder of the client terminal in the subfolder MQL4\files (or Tester\Files in case of testing). If FILE_COMMON is specified among flags, the file is opened in a shared folder for all MetaTrader 4 client terminals.

"Named pipes" can be opened according to the following rules:

- Pipe name is a string, which should have the following look: "\\servername\pipe\pipename", where servername - server name in the network, while pipename is a pipe name. If the pipes are used on the same computer, the server name can be omitted but a point should be inserted instead of it: "\\.\pipe\pipename". A client trying to connect the pipe should know its name.
- FileFlush() and FileSeek() should be called to the beginning of a file between sequential operations of reading from the pipe and writing to it.

A special symbol '\' is used in shown strings. Therefore, '\' should be doubled when writing a name in MQL4 application. It means that the above example should have the following look in the code: "\\\\servername\\pipe\\pipename".

More information about working with named pipes can be found in the article "Communicating With MetaTrader 5 Using Named Pipes Without Using DLLs".

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- incorrect file opening method
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
   string filename=terminal_data_path+"\\MQL4\\Files\\"+"fractals.csv";
   int filehandle=FileOpen(filename,FILE_WRITE|FILE_CSV);
   if(filehandle<0)
     {
      Print("Failed to open the file by the absolute path ");
      Print("Error code ",GetLastError());
     }
//--- correct way of working in the "file sandbox"
   ResetLastError();
   filehandle=FileOpen("fractals.csv",FILE_WRITE|FILE_CSV);
   if(filehandle!=INVALID_HANDLE)
     {
      FileWrite(filehandle,TimeCurrent(),Symbol(), EnumToString(ENUM_TIMEF
      FileClose(filehandle);
      Print("FileOpen OK");
     }
   else Print("Operation FileOpen failed, error ",GetLastError());
//--- another example with the creation of an enclosed directory in MQL4\F
   string subfolder="Research";
   filehandle=FileOpen(subfolder+"\\fractals.txt",FILE_WRITE|FILE_CSV);
      if(filehandle!=INVALID_HANDLE)
     {
      FileWrite(filehandle,TimeCurrent(),Symbol(), EnumToString(ENUM_TIMEF
      FileClose(filehandle);
      Print("The file most be created in the folder "+terminal_data_path+"
     }
   else Print("File open failed, error ",GetLastError());
  }
```

See also

[Use of a Codepage](), [FileFindFirst()](), [FolderCreate()](), [File opening flags]()

FileClose

Close the file previously opened by FileOpen().

```
void  FileClose(    int  file_handle      // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

No value returned.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";   // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- print the path to the file we are going to use
   PrintFormat("Working %s\\Files\\ folder",TerminalInfoString(TERMINAL_DA
//--- reset the error value
   ResetLastError();
//--- open the file for reading (if the file does not exist, the error wil
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      //--- print the file contents
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      //--- close the file
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, code = %d",GetLastError());
  }
```

See also

FileOpen()

FileCopy

The function copies the original file from a local or shared folder to another file.

```
bool  FileCopy(    const string  src_file_name,      // Name of a source fi
   int              common_flag,         // Location
   const string  dst_file_name,         // Name of the destination file
   int              mode_flags           // Access mode
   );
```

Parameters

src_file_name

[in] File name to copy.

common_flag

[in] Flag determining the location of the file. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder (for example, *common_flag=0*).

dst_file_name

[in] Result file name.

mode_flags

[in] Access flags. The parameter can contain only 2 flags: FILE_REWRITE and/or FILE_COMMON - other flags are ignored. If the file already exists, and the FILE_REWRITE flag hasn't been specified, then the file will not be rewritten, and the function will return false.

Return Value

In case of failure the function returns false.

Note

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means, cannot be outside the file sandbox.

If the new file already exists, the copy will be made depending on the availability of the FILE_REWRITE flag in the mode_flags parameter.

Example:

```
//--- display the window of input parameters when launching the script
```

```
#property script_show_inputs
//--- input parameters
input string InpSrc="source.txt";       // source
input string InpDst="destination.txt";  // copy
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- display the source contents (it must exist)
   if(!FileDisplay(InpSrc))
      return;
//--- check if the copy file already exists (may not be created)
   if(!FileDisplay(InpDst))
     {
      //--- the copy file does not exist, copying without FILE_REWRITE fla
      if(FileCopy(InpSrc,0,InpDst,0))
         Print("File is copied!");
      else
         Print("File is not copied!");
     }
   else
     {
      //--- the copy file already exists, try to copy without FILE_REWRITE
      if(FileCopy(InpSrc,0,InpDst,0))
         Print("File is copied!");
      else
         Print("File is not copied!");
      //--- InpDst file's contents remains the same
      FileDisplay(InpDst);
      //--- copy once more with FILE_REWRITE flag (correct copying if the
      if(FileCopy(InpSrc,0,InpDst,FILE_REWRITE))
         Print("File is copied!");
      else
         Print("File is not copied!");
     }
//--- receive InpSrc file copy
   FileDisplay(InpDst);
  }
//+------------------------------------------------------------------+
//| Read the file contents                                           |
//+------------------------------------------------------------------+
bool FileDisplay(const string file_name)
  {
//--- reset the error value
   ResetLastError();
//--- open the file
   int file_handle=FileOpen(file_name,FILE_READ|FILE_TXT|InpEncodingType);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- display the file contents in the loop
      Print("+---------------------+");
      PrintFormat("File name = %s",file_name);
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      Print("+---------------------+");
      //--- close the file
      FileClose(file_handle);
      return(true);
     }
//--- failed to open the file
   PrintFormat("%s is not opened, error = %d",file_name,GetLastError());
   return(false);
  }
```

FileDelete

Deletes the specified file in a local folder of the client terminal.

```
bool  FileDelete(    const string  file_name,      // File name to delete
   int            common_flag=0   // Location of the file to delete
   );
```

Parameters

file_name

[in] File name.

common_flag=0

[in] Flag determining the file location. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder.

Return Value

In case of failure the function returns false.

Note

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means, cannot be outside the file sandbox.

Deletes the specified file from a local folder of the client terminal (MQL4\Files or MQL4\Tester\Files in case of testing). If common_flag = FILE_COMMON, then the function removes the file from the shared folder for all client terminals.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- date for old files
input datetime InpFilesDate=D'2013.01.01 00:00';
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // variable for storing file names
   string   filter="*.txt"; // filter for searching the files
   datetime create_date;    // file creation date
   string   files[];        // list of file names
   int         def_size=25;    // array size by default
   int         size=0;         // number of files
//--- allocate memory for the array
   ArrayResize(files,def_size);
//--- receive the search handle in the local folder's root
   long search_handle=FileFindFirst(filter,file_name);
//--- check if FileFindFirst() executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- searching files in the loop
      do
        {
         files[size]=file_name;
         //--- increase the array size
         size++;
         if(size==def_size)
           {
            def_size+=25;
            ArrayResize(files,def_size);
           }
         //--- reset the error value
         ResetLastError();
         //--- receive the file creation date
         create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,f
         //--- check if the file is old
         if(create_date<InpFilesDate)
           {
            PrintFormat("%s file deleted!",file_name);
            //--- delete the old file
            FileDelete(file_name);
           }
        }
      while(FileFindNext(search_handle,file_name));
      //--- close the search handle
      FileFindClose(search_handle);
     }
   else
     {
      Print("Files not found!");
      return;
     }
//--- check what files have remained
   PrintFormat("Results:");
   for(int i=0;i<size;i++)
     {
      if(FileIsExist(files[i]))
         PrintFormat("%s file exists!",files[i]);
      else
         PrintFormat("%s file deleted!",files[i]);
     }
  }
```

FileMove

Moves a file from a local or shared folder to another folder.

```
bool   FileMove(    const string   src_file_name,     // File name for the mo
   int              common_flag,       // Location
   const string     dst_file_name,     // Name of the destination file
   int              mode_flags         // Access mode
   );
```

Parameters

src_file_name

[in] File name to move/rename.

common_flag

[in] Flag determining the location of the file. If common_flag = FILE_COMMON, then the file is located in a shared folder for all client terminals \Terminal\Common\Files. Otherwise, the file is located in a local folder (*common_flag=0*).

dst_file_name

[in] File name after operation

mode_flags

[in] Access flags. The parameter can contain only 2 flags: FILE_REWRITE and/or FILE_COMMON - other flags are ignored. If the file already exists and the FILE_REWRITE flag isn't specified, the file will not be rewritten, and the function will return false.

Return Value

In case of failure the function returns false.

Note

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means, cannot be outside the file sandbox.

If the new file already exists, the copy will be made depending on the availability of the FILE_REWRITE flag in the mode_flags parameter.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
```

```
input string InpSrcName="data.txt";
input string InpDstName="newdata.txt";
input string InpSrcDirectory="SomeFolder";
input string InpDstDirectory="OtherFolder";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string local=TerminalInfoString(TERMINAL_DATA_PATH);
   string common=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
//--- get file paths
   string src_path=InpSrcDirectory+"//"+InpSrcName;
   string dst_path=InpDstDirectory+"//"+InpDstName;
//--- check if the source file exists (if not - exit)
   if(FileIsExist(src_path))
      PrintFormat("%s file exists in the %s\\Files\\%s folder",InpSrcName,
   else
     {
      PrintFormat("Error, %s source file not found",InpSrcName);
      return;
     }
//--- check if the result file already exists
   if(FileIsExist(dst_path,FILE_COMMON))
     {
      PrintFormat("%s file exists in the %s\\Files\\%s folder",InpDstName,
      //--- file exists, moving should be performed with FILE_REWRITE flag
      ResetLastError();
      if(FileMove(src_path,0,dst_path,FILE_COMMON|FILE_REWRITE))
         PrintFormat("%s file moved",InpSrcName);
      else
         PrintFormat("Error! Code = %d",GetLastError());
     }
   else
     {
      PrintFormat("%s file does not exist in the %s\\Files\\%s folder",Inp
      //--- the file does not exist, moving should be performed without FI
      ResetLastError();
      if(FileMove(src_path,0,dst_path,FILE_COMMON))
         PrintFormat("%s file moved",InpSrcName);
      else
         PrintFormat("Error! Code = %d",GetLastError());
     }
//--- the file is moved; let's check it out
   if(FileIsExist(dst_path,FILE_COMMON) && !FileIsExist(src_path,0))
      Print("Success!");
   else
```

```
         Print("Error!");
   }
```

See also

FileIsExist()

FileFlush

Writes to a disk all data remaining in the input/output file buffer.

```
void  FileFlush(    int   file_handle       // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

No value returned.

Note

When writing to a file, the data may be actually found there only after some time. To save the data in the file instantly, use FileFlush() function. If the function is not used, part of the data that has not been stored in the disk yet, will be forcibly written there only when the file is closed using FileClose() function.

The function should be used when written data is of a certain value. It should be kept in mind that frequent function call may affect the program operation speed.

Function FileFlush () must be called between the operations of reading from a file and writing to it.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- file name for writing
input string InpFileName="example.csv"; // file name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- reset error value
   ResetLastError();
//--- open the file
   int file_handle=FileOpen(InpFileName,FILE_READ|FILE_WRITE|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- write data to the file
      for(int i=0;i<1000;i++)
        {
         //--- call write function
         FileWrite(file_handle,TimeCurrent(),SymbolInfoDouble(Symbol(),SYM
         //--- save data on the disk at each 128th iteration
         if((i & 127)==127)
           {
            //--- now, data will be located in the file and will not be lo
            FileFlush(file_handle);
            PrintFormat("i = %d, OK",i);
           }
         //--- 0.01 second pause
         Sleep(10);
        }
      //--- close the file
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, code = %d",GetLastError());
  }
```

See also

FileClose()

FileGetInteger

Gets an integer property of a file. There are two variants of the function.

1. Get a property by the handle of a file.

```
long  FileGetInteger(    int                          file_handle,    // Fil
   ENUM_FILE_PROPERTY_INTEGER  property_id    // Property ID
   );
```

2. Get a property by the file name.

```
long  FileGetInteger(
   const string                 file_name,             // File name
   ENUM_FILE_PROPERTY_INTEGER  property_id,           // Property ID
   bool                         common_folder=false    // The file is viewed
   );                                                  // or a common folder
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

file_name

[in] File name.

property_id

[in] File property ID. The value can be one of the values of the ENUM_FILE_PROPERTY_INTEGER enumeration. If the second variant of the function is used, you can receive only the values of the following properties: FILE_EXISTS, FILE_CREATE_DATE, FILE_MODIFY_DATE, FILE_ACCESS_DATE and FILE_SIZE.

common_folder=false

[in] Points to the file location. If the parameter is false, terminal data folder is viewed. Otherwise it is assumed that the file is in the shared folder of all terminals \Terminal\Common\Files (FILE_COMMON).

Return Value

The value of the property. In case of an error, -1 is returned. To get an error code use the GetLastError() function.

If a folder is specified when getting properties by the name, the function will have error 5018 (ERR_MQL_FILE_IS_DIRECTORY) in any case, though the return value will be correct.

Note

The function always changes the error code. In case of successful completion the error code is reset to NULL.

Example:

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="data.csv";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string path=InpDirectoryName+"//"+InpFileName;
   long   l=0;
//--- open the file
   ResetLastError();
   int handle=FileOpen(path,FILE_READ|FILE_CSV);
   if(handle!=INVALID_HANDLE)
     {
      //--- print all information about the file
      Print(InpFileName," file info:");
      FileInfo(handle,FILE_EXISTS,l,"bool");
      FileInfo(handle,FILE_CREATE_DATE,l,"date");
      FileInfo(handle,FILE_MODIFY_DATE,l,"date");
      FileInfo(handle,FILE_ACCESS_DATE,l,"date");
      FileInfo(handle,FILE_SIZE,l,"other");
      FileInfo(handle,FILE_POSITION,l,"other");
      FileInfo(handle,FILE_END,l,"bool");
      FileInfo(handle,FILE_IS_COMMON,l,"bool");
      FileInfo(handle,FILE_IS_TEXT,l,"bool");
      FileInfo(handle,FILE_IS_BINARY,l,"bool");
      FileInfo(handle,FILE_IS_CSV,l,"bool");
      FileInfo(handle,FILE_IS_ANSI,l,"bool");
      FileInfo(handle,FILE_IS_READABLE,l,"bool");
      FileInfo(handle,FILE_IS_WRITABLE,l,"bool");
      //--- close the file
      FileClose(handle);
     }
   else
      PrintFormat("%s file is not opened, ErrorCode = %d",InpFileName,GetI
  }
//+------------------------------------------------------------------+
//| Display the value of the file property                           |
//+------------------------------------------------------------------+
void FileInfo(const int handle,const ENUM_FILE_PROPERTY_INTEGER id,
              long l,const string type)
  {
//--- receive the property value
   ResetLastError();
   if((l=FileGetInteger(handle,id))!=-1)
     {
      //--- the value received, display it in the correct format
      if(!StringCompare(type,"bool"))
         Print(EnumToString(id)," = ",l ? "true" : "false");
      if(!StringCompare(type,"date"))
         Print(EnumToString(id)," = ",(datetime)l);
      if(!StringCompare(type,"other"))
         Print(EnumToString(id)," = ",l);
     }
   else
      Print("Error, Code = ",GetLastError());
  }
```

See also

File Operations, File Properties

FileIsEnding

Defines the end of a file in the process of reading.

```
bool  FileIsEnding(    int  file_handle      // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The function returns true if the file end has been reached in the process of reading or moving of the file pointer.

Note

To define the end of the file, the function tries to read the next string from it. If the string does not exist, the function returns true, otherwise it returns false.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";   // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- print the path to the file we are going to use
   PrintFormat("Working %s\\Files\\ folder",TerminalInfoString(TERMINAL_DA
//--- reset the error value
   ResetLastError();
//--- open the file for reading (if the file does not exist, the error wil
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      //--- print the file contents
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      //--- close the file
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, code = %d",GetLastError());
  }
```

FileIsLineEnding

Defines the line end in a text file in the process of reading.

```
bool  FileIsLineEnding(    int  file_handle      // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

Returns true if in the process of reading txt or csv-file reached the end of the line (the characters CR-LF).

Example (the file obtained during the execution of an example for FileWriteString() function is used here)

```
//+------------------------------------------------------------------+
//|                                       Demo_FileIsLineEnding.mq5 |
//|                        Copyright 2014, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- parameters for data reading
input string InpFileName="RSI.csv";   // file name
input string InpDirectoryName="Data"; // directory name
//--- overbought variables
int      ovb_size=0;
datetime ovb_time[];
//--- oversold variables
int      ovs_size=0;
datetime ovs_time[];
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      double value;
      //--- read data from file
      while(!FileIsEnding(file_handle))
        {
         //--- read the first value in the string
         value=FileReadNumber(file_handle);
         //--- read to different arrays according to the function result
         if(value>=70)
            ReadData(file_handle,ovb_time,ovb_size);
         else
            ReadData(file_handle,ovs_time,ovs_size);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
      //--- print data
      PrintFormat("Overbought=%d",ovb_size);
      for(int i=0; i<ovb_size; i++) Print(i," time=",TimeToString(ovb_time
      PrintFormat("Oversold=%d",ovs_size);
      for(int i=0; i<ovs_size; i++) Print(i," time=",TimeToString(ovs_time
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
      return;
     }
//---
  }
//+------------------------------------------------------------------+
//| Read the file's string data                                      |
//+------------------------------------------------------------------+
void ReadData(const int file_handle,datetime &arr[],int &size)
  {
   bool flag=false;
   string str="";
//--- read till the end of the string or of the file is reached
   while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
     {
      //--- shift the position by reading the number
      if(flag)
         FileReadNumber(file_handle);
      size++;
      //--- increase the array size if necessary
      if(size>ArraySize(arr)) ArrayResize(arr,size,100);
      //--- read date
      arr[size-1]=FileReadDatetime(file_handle);
      //--- add to string
      str+=" "+TimeToString(arr[size-1]);
      //--- slip past the first iteration
      flag=true;
     }
   Print(str);
  }
```

See also

FileWriteString()

FileReadArray

Reads from a file of BIN type arrays of any type except string (may be an array of structures, not containing strings, and dynamic arrays).

```
uint  FileReadArray(   int     file_handle,              // File handle
   void&  array[],                   // Array to record
   int    start=0,                   // start array position to write
   int    count=WHOLE_ARRAY          // count to read
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

array[]

[out] An array where the data will be loaded.

start=0

[in] Start position to read from the array.

count=WHOLE_ARRAY

[in] Number of elements to read. By default, reads the entire array (count=WHOLE_ARRAY).

Return Value

Number of elements read or 0 in case of error. To obtain information about the error, call the GetLastError() function. By default the count=WHOLE_ARRAY elements will be read.

Note

String array can be read only from the file of TXT type. If necessary, the function tries to increase the size of the array.

Example (the file obtained after execution of the example for FileWriteArray() function is used here)

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| Structure for storing price data                                 |
//+------------------------------------------------------------------+
struct prices
  {
   datetime          date; // date
   double            bid;  // bid price
   double            ask;  // ask price
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- structure array
   prices arr[];
//--- file path
   string path=InpDirectoryName+"//"+InpFileName;
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(path,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- read all data from the file to the array
      FileReadArray(file_handle,arr);
      //--- receive the array size
      int size=ArraySize(arr);
      //--- print data from the array
      for(int i=0;i<size;i++)
         Print("Date = ",arr[i].date," Bid = ",arr[i].bid," Ask = ",arr[i]
      Print("Total data = ",size);
      //--- close the file
      FileClose(file_handle);
     }
   else
      Print("File open failed, error ",GetLastError());
  }
```

See also

Variables, FileWriteArray()

FileReadBool

Reads from the file of CSV type string from the current position to a delimiter (or till the end of the text line) and converts the read string to a bool type value.

```
bool  FileReadBool(    int  file_handle    // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

Line read may be set to "true", "false" or the symbolic representation of integers "0" or "1". A nonzero value is converted to a logical true. The function returns the converted value.

Example (the file obtained after executing the example for FileWrite() function is used here)

```
//+------------------------------------------------------------------+
//|                                            Demo_FileReadBool.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
//---- plot Label1
#property indicator_label1  "UpSignal"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- plot Label2
#property indicator_label2  "DownSignal"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- parameters for data reading
input string InpFileName="MACD.csv";  // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int      ind=0;       // index
double   upbuff[];    // indicator buffers of up arrows
double   downbuff[];  // indicator buffer of down arrows
bool     sign_buff[]; // signal array (true - buy, false - sell)
datetime time_buff[]; // array of signals' arrival time
int      size=0;      // size of signal arrays
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for reading",InpFileName);
      //--- first, read the number of signals
      size=(int)FileReadNumber(file_handle);
      //--- allocate memory for the arrays
      ArrayResize(sign_buff,size);
      ArrayResize(time_buff,size);
      //--- read data from the file
      for(int i=0;i<size;i++)
        {
         //--- signal time
         time_buff[i]=FileReadDatetime(file_handle);
         //--- signal value
         sign_buff[i]=FileReadBool(file_handle);
        }
      //--- close the file
      FileClose(file_handle);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
      return(INIT_FAILED);
     }
//--- binding the arrays
   SetIndexBuffer(0,upbuff,INDICATOR_DATA);
   SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- set the symbol code for drawing in PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,241);
   PlotIndexSetInteger(1,PLOT_ARROW,242);
//---- set the indicator values that will not be seen on the chart
```

```
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
   PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(high,false);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      upbuff[i]=0;
      downbuff[i]=0;
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if dates coincide, use the value from the file
            if(time[i]==time_buff[j])
              {
               //--- draw the arrow according to the signal
               if(sign_buff[j])
                  upbuff[i]=high[i];
               else
                  downbuff[i]=low[i];
               //--- increase the counter
               ind=j+1;
               break;
              }
           }
        }
      }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Type bool, FileWrite()

FileReadDatetime

Reads from the file of CSV type a string of one of the formats: "YYYY.MM.DD HH:MI:SS", "YYYY.MM.DD" or "HH:MI:SS" - and converts it into a value of datetime type.

```
datetime  FileReadDatetime(    int  file_handle    // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of datetime type.

Example (the file obtained after executing the example for FileWrite() function is used here)

```
//+------------------------------------------------------------------+
//|                                        Demo_FileReadDateTime.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
//---- plot Label1
#property indicator_label1  "UpSignal"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  4
//---- plot Label2
#property indicator_label2  "DownSignal"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  4
//--- parameters for data reading
input string InpFileName="MACD.csv";  // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
```

```
int      ind=0;       // index
double   upbuff[];    // indicator buffers of up arrows
double   downbuff[];  // indicator buffer of down arrows
bool     sign_buff[]; // signal array (true - buy, false - sell)
datetime time_buff[]; // array of signals' arrival time
int      size=0;      // size of signal arrays
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for reading",InpFileName);
      //--- first, read the number of signals
      size=(int)FileReadNumber(file_handle);
      //--- allocate memory for the arrays
      ArrayResize(sign_buff,size);
      ArrayResize(time_buff,size);
      //--- read data from the file
      for(int i=0;i<size;i++)
        {
         //--- signal time
         time_buff[i]=FileReadDatetime(file_handle);
         //--- signal value
         sign_buff[i]=FileReadBool(file_handle);
        }
      //--- close the file
      FileClose(file_handle);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
      return(INIT_FAILED);
     }
//--- binding the arrays
   SetIndexBuffer(0,upbuff,INDICATOR_DATA);
   SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- set the symbol code for drawing in PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,241);
   PlotIndexSetInteger(1,PLOT_ARROW,242);
//---- set the indicator values that will not be seen on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
   PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0);
```

```
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(high,false);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      upbuff[i]=0;
      downbuff[i]=0;
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if dates coincide, use the value from the file
            if(time[i]==time_buff[j])
              {
               //--- draw the arrow according to the signal
               if(sign_buff[j])
                  upbuff[i]=high[i];
               else
                  downbuff[i]=low[i];
               //--- increase the counter
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
```

```
   return(rates_total);
  }
```

See also

Type datetime, StringToTime(), TimeToString(), FileWrite()

FileReadDouble

Reads a double-precision floating point number (double) from the current position of the binary file.

```
double  FileReadDouble(    int   file_handle,        // File handle
   int   size=DOUBLE_VALUE   // Size
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

size=DOUBLE_VALUE

Number of bytes (up to 8 inclusive), that should be read. The corresponding constants are provided: DOUBLE_VALUE = 8, FLOAT_VALUE = 4, so the function can read the whole value of double or float type.

Return Value

The value of double type.

Note

For more details about the error, call GetLastError().

Example (the file obtained after executing the example for FileWriteDouble() function is used here)

```
//+------------------------------------------------------------------+
//|                                      Demo_FileReadDouble.mq4 |
//|                        Copyright 2014, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
#property indicator_chart_window
#property indicator_buffers 1
//---- plot Label1
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_separate_window
```

```
//--- data reading parameters
input string InpFileName="MA.bin";     // File name
input string InpDirectoryName="Data"; // Folder name
//--- global variables
int      ind=0;
int      size=0;
double   ma_buff[];
datetime time_buff[];
//--- indicator buffer
double   buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- first, read the amount of data in the file
      size=(int)FileReadDouble(file_handle,DOUBLE_VALUE);
      //--- allocate memory for the arrays
      ArrayResize(ma_buff,size);
      ArrayResize(time_buff,size);
      //--- read data from the file
      for(int i=0;i<size;i++)
        {
         time_buff[i]=(datetime)FileReadDouble(file_handle,DOUBLE_VALUE);
         ma_buff[i]=FileReadDouble(file_handle,DOUBLE_VALUE);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
      return(INIT_FAILED);
     }
//--- bind the array to the indicator buffer with index 0
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- set the indicator values that will not be visible on the chart
   SetIndexEmptyValue(0,0.0);
//--- set indexing as timeseries
```

```
   ArraySetAsSeries(time_buff,true);
   ArraySetAsSeries(ma_buff,true);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,true);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      buff[i]=0;
      for(int j=0;j<size;j++)
        {
         //--- if the dates coincide, the value from the file is used
         if(time[i]==time_buff[j])
           {
            buff[i]=ma_buff[j];
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Real types (double, float), StringToDouble(), DoubleToString(), FileWriteDouble()

FileReadFloat

Reads the single-precision floating point number (float) from the current position of the binary file.

```
float  FileReadFloat(    int  file_handle     // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of float type.

Note

For more details about the error, call GetLastError().

Example (the file obtained after executing the example for FileWriteFloat() function is used here)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadFloat.mq4 |
//|                        Copyright 2014, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
//--- parameters for data reading
input string InpFileName="Close.bin"; // file name
input string InpDirectoryName="Data"; // directory name
//--- global variables
int      size=0;
double   close_buff[];
datetime time_buff[];
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- read data from the file
      while(!FileIsEnding(file_handle))
        {
         size++;
         //--- check size
         if(size>ArraySize(time_buff)) ArrayResize(time_buff,size,100);
         if(size>ArraySize(close_buff)) ArrayResize(close_buff,size,100);
         //--- read time and price values
         time_buff[size-1]=(datetime)FileReadDouble(file_handle);
         close_buff[size-1]=(double)FileReadFloat(file_handle);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
      //--- check arrays
      if(ArraySize(time_buff)==ArraySize(close_buff))
        {
         //--- print data
         PrintFormat("Read data:%d",ArraySize(time_buff));
         for(int i=0; i<ArraySize(time_buff); i++) PrintFormat("%d, time=%
        }
        else
        Print("Error in data.");
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
      return;
     }
  }
```

See also

Real types (double, float), FileReadDouble(), FileWriteFloat()

FileReadInteger

The function reads int, short or char value from the current position of the file pointer depending on the length specified in bytes.

```
int  FileReadInteger(    int  file_handle,        // File handle
   int  size=INT_VALUE      // Size of an integer in bytes
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

size=INT_VALUE

[in] Number of bytes (up to 4 inclusive) that should be read. The corresponding constants are provided: CHAR_VALUE = 1, SHORT_VALUE = 2 and INT_VALUE (LONG_VALUE) = 4, so the function can read the whole value of char, short, int or long type.

Return Value

A value of the int type. The result of this function must be explicitly cast to a target type, i.e. to the type of data that you need to read. Since a value of the int type is returned, it can be easily converted to any integer value. The file pointer is shifted by the number of bytes read.

Note

When reading less than 4 bytes, the received result is always positive. If one or two bytes are read, the sign of the number can be determined by explicit casting to type char (1 byte) or short (2 bytes). Getting the sign for a three-byte number is not trivial, since there is no corresponding underlying type.

Example (the file obtained after executing the example for FileWriteInteger() function is used here)

```
//+------------------------------------------------------------------+
//|                                         Demo_FileReadInteger.mq4 |
//|                        Copyright 2014, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
#property indicator_chart_window
```

```
#property indicator_buffers 1
//---- plot Label1
#property indicator_label1  "Trends"
#property indicator_type1   DRAW_SECTION
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parameters for data reading
input string InpFileName="Trend.bin"; // File name
input string InpDirectoryName="Data"; // Folder name
//--- global variables
int      ind=0;
int      size=0;
datetime time_buff[];
//--- indicator buffers
double   buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
   int def_size=100;
//--- allocate memory for the array
   ArrayResize(time_buff,def_size);
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- additional variables
      int    arr_size;
      uchar  arr[];
      //--- read data from the file
      while(!FileIsEnding(file_handle))
        {
         //--- find out how many bytes are used for writing the time
         arr_size=FileReadInteger(file_handle,INT_VALUE);
         ArrayResize(arr,arr_size);
         for(int i=0;i<arr_size;i++)
            arr[i]=(char)FileReadInteger(file_handle,CHAR_VALUE);
         //--- store the time value
         time_buff[size]=StringToTime(CharArrayToString(arr));
         size++;
         //--- increase the sizes of the arrays if they are filled
         if(size==def_size)
```

```
           {
            def_size+=100;
            ArrayResize(time_buff,def_size);
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
      return(INIT_FAILED);
     }
//--- bind the array to the indicator buffer
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- set the indicator values that will not be visible on the chart
   SetIndexEmptyValue(0,0.0);
//--- set indexing as timeseries
   ArraySetAsSeries(time_buff,true);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,true);
   ArraySetAsSeries(close,true);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      buff[i]=0;
      for(int j=0;j<size;j++)
        {
         //--- if dates coincide, set as close price
         if(time[i]==time_buff[j])
           {
            //--- set as close price
            buff[i]=close[i];
           }
        }
     }
//---
   return(rates_total);
  }
```

See also

IntegerToString(), StringToInteger(), Integer types, FileWriteInteger()

FileReadLong

The function reads an integer of long type (8 bytes) from the current position of the binary file.

```
long  FileReadLong(    int   file_handle     // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of long type.

Example (the file obtained during the execution of an example for FileWriteLong() function is used here)

```
//+------------------------------------------------------------------+
//|                                            Demo_FileReadLong.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_buffers 1
//---- plot Label1
#property indicator_label1  "Volume"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrYellow
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- parameters for data reading
input string InpFileName="Volume.bin"; // file name
input string InpDirectoryName="Data";  // directory name
//--- global variables
int      ind=0;
int      size=0;
long     volume_buff[];
datetime time_buff[];
//--- indicator buffers
double   buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- first, read the amount of data in the file
      size=(int)FileReadLong(file_handle);
      //--- allocate memory for the arrays
      ArrayResize(volume_buff,size);
      ArrayResize(time_buff,size);
      //--- read data from the file
      for(int i=0;i<size;i++)
        {
         time_buff[i]=(datetime)FileReadLong(file_handle);
         volume_buff[i]=FileReadLong(file_handle);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
      return(INIT_FAILED);
     }
//--- bind the array to the indicator buffer with 0 index
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- set the indicator values that will be visible on the chart
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
//--- the loop for the bars that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      buff[i]=0;
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if dates coincide, the value from the file is used
            if(time[i]==time_buff[j])
              {
               buff[i]=(double)volume_buff[j];
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Integer types, FileReadInteger(), FileWriteLong()

FileReadNumber

The function reads from the CSV file a string from the current position till a separator (or till the end of a text string) and converts the read string to a value of double type.

```
double  FileReadNumber(    int   file_handle     // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of double type.

Example script in FileIsLineEnding().

See also

FileWriteString()

FileReadString

The function reads a string from the current position of a file pointer in a file.

```
string  FileReadString(    int   file_handle,      // File handle
   int  length=0          // Length of the string
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

length=0

[in] Number of characters to read.

Return Value

Line read (string).

Note

When reading from a bin-file. the length of a string to read must be specified. When reading from a txt-file the string length is not required, and the string will be read from the current position to the line feed character "\r\n". When reading from a csv-file, the string length isn't required also, the string will be read from the current position till the nearest delimiter or till the text string end character.

If the file is opened with FILE_ANSI flag, then the line read is converted to Unicode.

Example (the file obtained after executing the example for FileWriteInteger() function is used here)

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for data reading
input string InpFileName="Trend.bin"; // file name
input string InpDirectoryName="Data"; // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- additional variables
      int    str_size;
      string str;
      //--- read data from the file
      while(!FileIsEnding(file_handle))
        {
         //--- find out how many symbols are used for writing the time
         str_size=FileReadInteger(file_handle,INT_VALUE);
         //--- read the string
         str=FileReadString(file_handle,str_size);
         //--- print the string
         PrintFormat(str);
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
  }
```

See also

String Type, Conversion Functions, FileWriteInteger()

FileReadStruct

The function reads contents into a structure passed as a parameter from a binary-file, starting with the current position of the file pointer.

```
uint   FileReadStruct(    int          file_handle,        // file handle
   const void&  struct_object,      // target structure to which the conte
   int          size=-1             // structure size in bytes
   );
```

Parameters

file_handle

[in] File descriptor of an open bin-file.

struct_object

[out] The object of this structure. The structure should not contain strings, dynamic arrays or virtual functions.

size=-1

[in] Number of bytes that should be read. If size is not specified or the indicated value is greater than the size of the structure, the exact size of the specified structure is used.

Return Value

If successful the function returns the number of bytes read or 0 in case of error. If successful, the number of bytes read corresponds to the size of the structure. To obtain information about the error call the GetLastError() function. File pointer is moved by the same number of bytes.

Example (the file obtained after using the example for FileWriteStruct() function is used here)

```
//+------------------------------------------------------------------+
//|                                       Demo_FileReadStruct.mq4 |
//|                      Copyright 2014, MetaQuotes Software Corp. |
//|                                           https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 4
//---- plot Label1
#property indicator_label1  "Open"
```

```
#property indicator_type1    DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_label1  "High"
#property indicator_type2    DRAW_LINE
#property indicator_color2  clrGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
#property indicator_label1  "Low"
#property indicator_type3    DRAW_LINE
#property indicator_color3  clrOrange
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
#property indicator_label1  "Close"
#property indicator_type4    DRAW_LINE
#property indicator_color4  clrRed
#property indicator_style4  STYLE_SOLID
#property indicator_width4  1
#property indicator_separate_window
//--- parameters for receiving data
input string  InpFileName="EURUSD.txt"; // file name
input string  InpDirectoryName="Data";  // directory name
//+------------------------------------------------------------------+
//| Structure for storing candlestick data                           |
//+------------------------------------------------------------------+
struct candlesticks
  {
   double            open;  // open price
   double            close; // close price
   double            high;  // high price
   double            low;   // low price
   datetime          date;  // date
  };
//--- indicator buffers
double       open_buff[];
double       close_buff[];
double       high_buff[];
double       low_buff[];
//--- global variables
candlesticks cand_buff[];
int          size=0;
int          ind=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
```

```
  {
   int default_size=100;
   ArrayResize(cand_buff,default_size);
//--- open the file
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_COM
      //--- read data from the file
      while(!FileIsEnding(file_handle))
        {
         //--- write data to the array
         uint bytesread=FileReadStruct(file_handle,cand_buff[size]);
         //--- check data read
         if (bytesread!=sizeof(candlesticks))
           {
            PrintFormat("Error reading data. Error code=%d",GetLastError()
            //--- close the file
            FileClose(file_handle);
            return(INIT_FAILED);
           }
         else
           {
            size++;
            //--- check if the array is overflown
            if(size==default_size)
              {
               //--- increase the array size
               default_size+=100;
               ArrayResize(cand_buff,default_size);
              }
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is read, %s file is closed",InpFileName);
     }
   else
     {
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
      return(INIT_FAILED);
     }
//--- indicator buffers mapping
   SetIndexBuffer(0,open_buff,INDICATOR_DATA);
   SetIndexBuffer(1,high_buff,INDICATOR_DATA);
   SetIndexBuffer(2,low_buff,INDICATOR_DATA);
   SetIndexBuffer(3,close_buff,INDICATOR_DATA);
//--- empty value
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   ArraySetAsSeries(time,false);
//--- the loop for the candlesticks that have not been handled yet
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- 0 by default
      open_buff[i]=0;
      close_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      //--- check if any data is still present
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- if dates coincide, the value from the file is used
            if(time[i]==cand_buff[j].date)
              {
               open_buff[i]=cand_buff[j].open;
               close_buff[i]=cand_buff[j].close;
               high_buff[i]=cand_buff[j].high;
               low_buff[i]=cand_buff[j].low;
               //--- increase the counter
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Structures and classes, FileWriteStruct()

FileSeek

The function moves the position of the file pointer by a specified number of bytes relative to the specified position.

```
bool  FileSeek(    int                  file_handle,     // File handle
   long                 offset,          // In bytes
   ENUM_FILE_POSITION   origin           // Position for reference
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

offset

[in] The shift in bytes (may take a negative value).

origin

[in] The starting point for the displacement. Can be one of values of ENUM_FILE_POSITION.

Return Value

If successful the function returns true, otherwise false. To obtain information about the error call the GetLastError() function.

Note

If the execution of the FileSeek() function results in a negative shift (going beyond the "level boundary" of the file), the file pointer will be set to the file beginning.

If a position is set beyond the "right boundary" of the file (larger than the file size), the next writing to the file will be performed not from the end of the file, but from the position set. In this case indefinite values will be written for the previous file end and the position set.

Example:

```
//+-------------------------------------------------------------------+
//|                                                 Demo_FileSeek.mq5 |
//|                         Copyright 2013, MetaQuotes Software Corp. |
//|                                               https://www.mql5.com |
//+-------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";    // file name
input string InpDirectoryName="Data";   // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- specify the value of the variable for generating random numbers
   _RandomSeed=GetTickCount();
//--- variables for positions of the strings' start points
   ulong pos[];
   int   size;
//--- reset the error value
   ResetLastError();
//--- open the file
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      //--- receive start position for each string in the file
      GetStringPositions(file_handle,pos);
      //--- define the number of strings in the file
      size=ArraySize(pos);
      if(!size)
        {
         //--- stop if the file does not have strings
         PrintFormat("%s file is empty!",InpFileName);
         FileClose(file_handle);
         return;
        }
      //--- make a random selection of a string number
      int ind=MathRand()%size;
      //--- shift position to the starting point of the string
      if(FileSeek(file_handle,pos[ind],SEEK_SET)==true)
        {
      //--- read and print the string with ind number
         PrintFormat("String text with %d number: \"%s\"",ind,FileReadStri
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("%s file is closed",InpFileName);
     }
   else
```

```
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
  }
//+-----------------------------------------------------------------------
//| The function defines starting points for each of the strings in the fi
//| places them in arr array
//+-----------------------------------------------------------------------
void GetStringPositions(const int handle,ulong &arr[])
  {
//--- default array size
   int def_size=127;
//--- allocate memory for the array
   ArrayResize(arr,def_size);
//--- string counter
   int i=0;
//--- if this is not the file's end, then there is at least one string
   if(!FileIsEnding(handle))
     {
      arr[i]=FileTell(handle);
      i++;
     }
   else
      return; // the file is empty, exit
//--- define the shift in bytes depending on encoding
   int shift;
   if(FileGetInteger(handle,FILE_IS_ANSI))
      shift=1;
   else
      shift=2;
//--- go through the strings in the loop
   while(1)
     {
      //--- read the string
      FileReadString(handle);
      //--- check for the file end
      if(!FileIsEnding(handle))
        {
         //--- store the next string's position
         arr[i]=FileTell(handle)+shift;
         i++;
         //--- increase the size of the array if it is overflown
         if(i==def_size)
           {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
           }
        }
      else
```

```
         break; // end of the file, exit
      }
//--- define the actual size of the array
   ArrayResize(arr,i);
  }
```

FileSize

The function returns the file size in bytes.

```
ulong  FileSize(    int  file_handle    // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

The value of type int.

Note

To obtain information about the error call GetLastError() function.

Example:

```
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input ulong  InpThresholdSize=20;        // file threshold size in kilobyt
input string InpBigFolderName="big";     // folder for large files
input string InpSmallFolderName="small"; // folder for small files
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // variable for storing file names
   string   filter="*.csv"; // filter for searching the files
   ulong    file_size=0;    // file size in bytes
   int      size=0;         // number of files
//--- print the path to the file we are going to work with
   PrintFormat("Working in %s\\Files\\ folder",TerminalInfoString(TERMINAL
//--- receive the search handle in common folder's root of all terminals
   long search_handle=FileFindFirst(filter,file_name,FILE_COMMON);
//--- check if FileFindFirst() has been executed successfully
   if(search_handle!=INVALID_HANDLE)
     {
      //--- move files in the loop according to their size
      do
        {
         //--- open the file
```

```
       ResetLastError();
       int file_handle=FileOpen(file_name,FILE_READ|FILE_CSV|FILE_COMMON
       if(file_handle!=INVALID_HANDLE)
         {
          //--- receive the file size
          file_size=FileSize(file_handle);
          //--- close the file
          FileClose(file_handle);
         }
       else
         {
          PrintFormat("Failed to open %s file, Error code = %d",file_nam
          continue;
         }
       //--- print the file size
       PrintFormat("Size of %s file is equal to %d bytes",file_name,file
       //--- define the path for moving the file
       string path;
       if(file_size>InpThresholdSize*1024)
          path=InpBigFolderName+"//"+file_name;
       else
          path=InpSmallFolderName+"//"+file_name;
       //--- move the file
       ResetLastError();
       if(FileMove(file_name,FILE_COMMON,path,FILE_REWRITE|FILE_COMMON))
          PrintFormat("%s file is moved",file_name);
       else
          PrintFormat("Error, code = %d",GetLastError());
      }
    while(FileFindNext(search_handle,file_name));
    //--- close the search handle
    FileFindClose(search_handle);
   }
 else
    Print("Files not found!");
}
```

FileTell

The file returns the current position of the file pointer of an open file.

```
ulong  FileTell(   int  file_handle    // File handle
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

Return Value

Current position of the file descriptor in bytes from the beginning of the file.

Note

To obtain information about the error call GetLastError().

Example:

```
//+------------------------------------------------------------------+
//|                                               Demo_FileTell.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- display the window of input parameters when launching the script
#property script_show_inputs
//--- input parameters
input string InpFileName="file.txt";     // file name
input string InpDirectoryName="Data";    // directory name
input int    InpEncodingType=FILE_ANSI; // ANSI=32 or UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- specify the value of the variable for generating random numbers
   _RandomSeed=GetTickCount();
//--- variables for positions of the strings' start points
   ulong pos[];
   int   size;
//--- reset the error value
```

```
   ResetLastError();
//--- open the file
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for reading",InpFileName);
      //--- receive start position for each string in the file
      GetStringPositions(file_handle,pos);
      //--- define the number of strings in the file
      size=ArraySize(pos);
      if(!size)
        {
         //--- stop if the file does not have strings
         PrintFormat("%s file is empty!",InpFileName);
         FileClose(file_handle);
         return;
        }
      //--- make a random selection of a string number
      int ind=MathRand()%size;
      //--- shift position to the starting point of the string
      FileSeek(file_handle,pos[ind],SEEK_SET);
      //--- read and print the string with ind number
      PrintFormat("String text with %d number: \"%s\"",ind,FileReadString(
      //--- close the file
      FileClose(file_handle);
      PrintFormat("%s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
  }
//+-----------------------------------------------------------------------
//| The function defines starting points for each of the strings in the fi
//| places them in arr array
//+-----------------------------------------------------------------------
void GetStringPositions(const int handle,ulong &arr[])
  {
//--- default array size
   int def_size=127;
//--- allocate memory for the array
   ArrayResize(arr,def_size);
//--- string counter
   int i=0;
//--- if this is not the file's end, then there is at least one string
   if(!FileIsEnding(handle))
     {
      arr[i]=FileTell(handle);
      i++;
     }
   else
      return; // the file is empty, exit
//--- define the shift in bytes depending on encoding
   int shift;
   if(FileGetInteger(handle,FILE_IS_ANSI))
      shift=1;
   else
      shift=2;
//--- go through the strings in the loop
   while(1)
     {
      //--- read the string
      FileReadString(handle);
      //--- check for the file end
      if(!FileIsEnding(handle))
        {
         //--- store the next string's position
         arr[i]=FileTell(handle)+shift;
         i++;
         //--- increase the size of the array if it is overflown
         if(i==def_size)
           {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
           }
        }
      else
         break; // end of the file, exit
     }
//--- define the actual size of the array
   ArrayResize(arr,i);
  }
```

FileWrite

The function is intended for writing of data into a CSV file, delimiter being inserted automatically unless it is equal to 0. After writing into the file, the line end character "\r\n" will be added.

```
uint  FileWrite(    int  file_handle,    // File handle
   ...                    // List of recorded parameters
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

...

[in] The list of parameters separated by commas, to write to the file. The number of written parameters can be up to 63.

Return Value

Number of bytes written or 0 in case of error. To obtain information about the error call the GetLastError() function.

Note

Numbers will be converted into a text at output (see the Print() function). Data of the double type are output with the accuracy of 16 digits after the decimal point, and the data can be displayed either in traditional or in scientific format - depending on which format will be the most compact. The data of the float type are shown with 5 digits after the decimal point. To output real numbers with different precision or in a clearly specified format, use DoubleToString().

Numbers of the bool type are displayed as "true" or "false" strings. Numbers of the datetime type are displayed as "YYYY.MM.DD HH:MI:SS".

Example:

```
//+------------------------------------------------------------------+
//|                                             Demo_FileWrite.mq4 |
//|                      Copyright 2014, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
```

```
#property strict
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string             InpSymbolName="EURUSD";      // Currency pair
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;   // Time frame
input int                InpFastEMAPeriod=12;         // Fast EMA period
input int                InpSlowEMAPeriod=26;         // Slow EMA period
input int                InpSignalPeriod=9;           // Difference averag
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // Price type
//--- parameters for writing data to file
input string             InpFileName="MACD.csv";      // File name
input string             InpDirectoryName="Data";     // Folder name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   bool     sign_buff[]; // signal array (true - buy, false - sell)
   datetime time_buff[]; // array of signals' appear time
   int      sign_size=0; // signal array size
   double   macd_buff[]; // array of indicator values
   datetime date_buff[]; // array of indicator dates
   int      macd_size=0; // size of indicator arrays
//--- set indexing as time series
   ArraySetAsSeries(sign_buff,true);
   ArraySetAsSeries(time_buff,true);
   ArraySetAsSeries(macd_buff,true);
   ArraySetAsSeries(date_buff,true);
//--- reset last error code
   ResetLastError();
//--- copying the time from last 1000 bars
   int copied=CopyTime(NULL,0,0,1000,date_buff);
   if(copied<=0)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastErr
      return;
     }
//--- prepare macd_buff array
   ArrayResize(macd_buff,copied);
//--- copy the values of main line of the iMACD indicator
   for(int i=0;i<copied;i++)
     {
      macd_buff[i]=iMACD(InpSymbolName,InpSymbolPeriod,InpFastEMAPeriod,In
     }
//--- get size
   macd_size=ArraySize(macd_buff);
```

```
//--- analyze the data and save the indicator signals to the arrays
   ArrayResize(sign_buff,macd_size-1);
   ArrayResize(time_buff,macd_size-1);
   for(int i=1;i<macd_size;i++)
     {
      //--- buy signal
      if(macd_buff[i-1]<0 && macd_buff[i]>=0)
        {
         sign_buff[sign_size]=true;
         time_buff[sign_size]=date_buff[i];
         sign_size++;
        }
      //--- sell signal
      if(macd_buff[i-1]>0 && macd_buff[i]<=0)
        {
         sign_buff[sign_size]=false;
         time_buff[sign_size]=date_buff[i];
         sign_size++;
        }
     }
//--- open the file for writing the indicator values (if the file is absen
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- first, write the number of signals
      FileWrite(file_handle,sign_size);
      //--- write the time and values of signals to the file
      for(int i=0;i<sign_size;i++)
         FileWrite(file_handle,time_buff[i],sign_buff[i]);
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
  }
```

See also

Comment(), Print(), StringFormat()

FileWriteArray

The function writes arrays of any type except for string to a BIN file (can be an array of structures not containing strings or dynamic arrays).

```
uint  FileWriteArray(   int            file_handle,         // File handle
   const void&  array[],             // Array
   int          start=0,             // Start index in the array
   int          count=WHOLE_ARRAY    // Number of elements
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

array[]

[out] Array for recording.

start=0

[in] Initial index in the array (number of the first recorded element).

count=WHOLE_ARRAY

[in] Number of items to write (WHOLE_ARRAY means all items starting with the number start until the end of the array).

Return Value

Number of elements written or 0 in case of error. To obtain information about the error call the GetLastError() function.

Note

String array can be written in a TXT file. In this case, strings are automatically ended by the line end characters "\r\n". Depending on the file type ANSI or UNICODE, strings are either converted to ansi-encoding or not.

Example:

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteArray.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- input parameters
```

```
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| Structure for storing price data                                 |
//+------------------------------------------------------------------+
struct prices
  {
   datetime          date; // date
   double            bid;  // bid price
   double            ask;  // ask price
  };
//--- global variables
int    count=0;
int    size=20;
string path=InpDirectoryName+"//"+InpFileName;
prices arr[];
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- allocate memory for the array
   ArrayResize(arr,size);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- write the remaining count strings if count<n
   WriteData(count);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- save data to array
   arr[count].date=TimeCurrent();
   arr[count].bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   arr[count].ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- show current data
   Print("Date = ",arr[count].date," Bid = ",arr[count].bid," Ask = ",arr[
//--- increase the counter
   count++;
//--- if the array is filled, write data to the file and zero it out
   if(count==size)
     {
      WriteData(size);
      count=0;
     }
  }
//+------------------------------------------------------------------+
//| Write n elements of the array to file                            |
//+------------------------------------------------------------------+
void WriteData(const int n)
  {
//--- open the file
   ResetLastError();
   int handle=FileOpen(path,FILE_READ|FILE_WRITE|FILE_BIN);
   if(handle!=INVALID_HANDLE)
     {
      //--- write array data to the end of the file
      FileSeek(handle,0,SEEK_END);
      FileWriteArray(handle,arr,0,n);
      //--- close the file
      FileClose(handle);
     }
   else
      Print("Failed to open the file, error ",GetLastError());
  }
```

See also

Variables, FileSeek()

FileWriteDouble

The function writes the value of a double parameter to a file, starting from the current position of the file pointer.

```
uint  FileWriteDouble(    int      file_handle,      // File handle
   double  value             // Value to write
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

value

[in] The value of double type.

Return Value

If successful the function returns the number of bytes written or 0 in case of error. If successful, the number of bytes written corresponds to the data type size (sizeof(double)=8). To obtain information about the error call the GetLastError() function. The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                         Demo_FileWriteDouble.mq4 |
//|                        Copyright 2014, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string             InpSymbolName="EUSRUSD";           // Currency pai
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;        // Time frame
input int                InpMAPeriod=10;                   // Smoothing pe
input int                InpMAShift=0;                     // Indicator sh
input ENUM_MA_METHOD     InpMAMethod=MODE_SMA;             // Smoothing ty
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;      // price type
//--- parameters for writing data to the file
input string             InpFileName="MA.bin";             // File name
input string             InpDirectoryName="Data";          // Folder name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   ma_buff[];
   datetime time_buff[];
   int      size;
//--- set indexing as timeseries
   ArraySetAsSeries(ma_buff,true);
   ArraySetAsSeries(time_buff,true);
//--- reset last erro
   ResetLastError();
//--- copying the time from last 1000 bars
   int copied=CopyTime(NULL,0,0,1000,time_buff);
   if(copied<=0)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastErr
      return;
     }
//--- prepare ma_buff[] array
   ArrayResize(ma_buff,copied);
//--- copy the values of iMA indicator
   for(int i=0;i<copied;i++)
     {
      ma_buff[i]=iMA(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpMAShift,
     }
//---
   PrintFormat("The values starting from %s to %s will be written to file.
//--- get size
   size=ArraySize(ma_buff);
//--- open the file for writing the indicator values (if the file is absen
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- first, write the size of data sample
      uint byteswritten=FileWriteDouble(file_handle,(double)size,DOUBLE_VA
      //--- check the number of bytes written
      if(byteswritten!=sizeof(double))
        {
         PrintFormat("Error in FileWriteDouble. Error code=%d",GetLastErr
         //--- close the file
         FileClose(file_handle);
         return;
        }
      //--- write the indicator time and value to the file
      for(int i=0;i<size;i++)
        {
         byteswritten=FileWriteDouble(file_handle,(double)time_buff[i],DOU
         //--- check the number of bytes written
         if(byteswritten!=sizeof(double))
           {
            PrintFormat("Error in FileWriteDouble. Error code=%d",GetLastE
            //--- close the file
            FileClose(file_handle);
            return;
           }
         byteswritten=FileWriteDouble(file_handle,ma_buff[i],DOUBLE_VALUE)
         //--- check number of bytes written
         if(byteswritten!=sizeof(double))
           {
            PrintFormat("Error in FileWriteDouble. Error code=%d",GetLastE
            //--- close the file
            FileClose(file_handle);
            return;
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
  }
```

See also

[Real types (double, float)]

FileWriteFloat

The function writes the value of the float parameter to a bin-file, starting from the current position of the file pointer.

```
uint  FileWriteFloat(    int     file_handle,      // File handle
   float  value              // Value to be written
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

value

[in] The value of float type.

Return Value

If successful the function returns the number of bytes written or 0 in case of error. If successful, the number of bytes written corresponds to the data type size (sizeof(float)=4). To obtain information about the error call the GetLastError() function. The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                         Demo_FileWriteFloat.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string          InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_M15;       // time frame
input datetime        InpDateStart=D'2013.01.01 00:00'; // data copying st
//--- parameters for writing data to the file
input string          InpFileName="Close.bin"; // file name
input string          InpDirectoryName="Data"; // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   close_buff[];
   datetime time_buff[];
   int      size;
//--- reset the error value
   ResetLastError();
//--- copy the close price for each bar
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,clo
     {
      PrintFormat("Failed to copy close price values. Error code = %d",Get
      return;
     }
//--- copy the time for each bar
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time
     {
      PrintFormat("Failed to copy the time values. Error code = %d",GetLas
      return;
     }
//--- receive the buffer size
   size=ArraySize(close_buff);
//--- open the file for writing the values (if the file is absent, it will
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- write close prices' time and values to the file
      for(int i=0;i<size;i++)
        {
         uint byteswritten=FileWriteDouble(file_handle,(double)time_buff[i
         //--- check the number of bytes written
         if(byteswritten!=sizeof(double))
           {
            PrintFormat("Error in FileWriteDouble. Error code=%d",GetLastE
            //--- close the file
            FileClose(file_handle);
            return;
           }
         byteswritten=FileWriteFloat(file_handle,(float)close_buff[i]);
         //--- check the number of bytes written
         if(byteswritten!=sizeof(float))
           {
            PrintFormat("Error in FileWriteDouble. Error code=%d",GetLastE
            //--- close the file
            FileClose(file_handle);
            return;
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
  }
```

See also

Real types (double, float), FileWriteDouble()

FileWriteInteger

The function writes the value of the int parameter to a bin-file, starting from the current position of the file pointer.

```
uint  FileWriteInteger(    int  file_handle,         // File handle
   int  value,                // Value to be written
   int  size=INT_VALUE        // Size in bytes
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

value

[in] Integer value.

size=INT_VALUE

[in] Number of bytes (up to 4 inclusive), that should be written. The corresponding constants are provided: CHAR_VALUE=1, SHORT_VALUE=2 and INT_VALUE=4, so the function can write the integer value of char, uchar, short, ushort, int, or uint type.

Return Value

If successful the function returns the number of bytes written or 0 in case of error. If successful, the number of bytes written corresponds to the data type size. To obtain information about the error call the GetLastError() function. The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                       Demo_FileWriteInteger.mq4 |
//|                          Copyright 2014, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string             InpSymbolName="EURUSD";      // Currency pair
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;  // Time frame
```

```
//--- parameters for writing data to the file
input string             InpFileName="Trend.bin";    // File name
input string             InpDirectoryName="Data";    // Folder name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   double   close_buff[];
   datetime time_buff[];
   int      size;
//--- set indexing as timeseries
   ArraySetAsSeries(close_buff,true);
   ArraySetAsSeries(time_buff,true);
//--- reset the error value
   ResetLastError();
//--- copy the close price for each bar
   if(CopyClose(InpSymbolName,InpSymbolPeriod,0,1000,close_buff)==-1)
     {
      PrintFormat("Failed to copy the values of close prices. Error code =
      return;
     }
//--- copy the time for each bar
   if(CopyTime(InpSymbolName,InpSymbolPeriod,0,1000,time_buff)==-1)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastErr
      return;
     }
//--- get the buffer size
   size=ArraySize(close_buff);
//--- open the file for writing the values (if the file is absent, it wil
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //---
      uint  byteswritten;// number of bytes written
      int   up_down=0;   // trend flag
      int   arr_size;    // arr array size
      uchar arr[];       // uchar type array
      //--- write time values to the file
      for(int i=0;i<size-1;i++)
        {
         //--- compare close prices of the current and next bars
         if(close_buff[i]<=close_buff[i+1])
           {
            if(up_down!=1)
              {
               //--- write date value to the file using FileWriteInteger
               StringToCharArray(TimeToString(time_buff[i]),arr);
               arr_size=ArraySize(arr);
               //--- first, write the number of chars in the array
               byteswritten=FileWriteInteger(file_handle,arr_size,INT_VALU
               //--- checking the number of bytes written
               if(byteswritten!=sizeof(int))
                 {
                  PrintFormat("Error in FileWriteInteger. Error code=%d",G
                  //--- close the file
                  FileClose(file_handle);
                  return;
                 }
               //--- write the chars
               for(int j=0;j<arr_size;j++)
                 {
                  byteswritten=FileWriteInteger(file_handle,arr[j],CHAR_VA
                  //--- checking the number of bytes written
                  if(byteswritten!=sizeof(char))
                    {
                     PrintFormat("Error in FileWriteInteger. Error code=%d
                     //--- close the file
                     FileClose(file_handle);
                     return;
                    }
                 }
               //--- change the trend flag
               up_down=1;
              }
           }
         else
           {
            if(up_down!=-1)
              {
               //--- write the date value to the file using FileWriteInteg
               StringToCharArray(TimeToString(time_buff[i]),arr);
               arr_size=ArraySize(arr);
               //--- first, write the number of chars in the array
               byteswritten=FileWriteInteger(file_handle,arr_size,INT_VALU
               //--- checking the number of bytes written
               if(byteswritten!=sizeof(int))
                 {
                  PrintFormat("Error in FileWriteInteger. Error code=%d",G
                  //--- close the file
```

```
                 FileClose(file_handle);
                 return;
                }
              //--- write chars
              for(int j=0;j<arr_size;j++)
                {
                 byteswritten=FileWriteInteger(file_handle,arr[j],CHAR_VA
                 //--- checking the number of bytes written
                 if(byteswritten!=sizeof(char))
                   {
                    PrintFormat("Error in FileWriteInteger. Error code=%d
                    //--- close the file
                    FileClose(file_handle);
                    return;
                   }
                }
              //--- change the trend flag
              up_down=-1;
             }
          }
       }
     //--- close the file
     FileClose(file_handle);
     PrintFormat("Data is written, %s file is closed",InpFileName);
    }
  else
     PrintFormat("Data is written, %s file is closed",InpFileName,GetLast
 }
```

See also

IntegerToString(), StringToInteger(), Integer types

FileWriteLong

The function writes the value of the long-type parameter to a bin-file, starting from the current position of the file pointer.

```
uint  FileWriteLong(    int    file_handle,      // File handle
   long   value             // Value to be written
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

value

[in] Value of type long.

Return Value

If successful the function returns the number of bytes written or 0 in case of error. If successful, the number of bytes written corresponds to the data type size (sizeof(long)=8). To obtain information about the error call the GetLastError() function. The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                        Demo_FileWriteLong.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string          InpSymbolName="EURUSD";           // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // time frame
input datetime        InpDateStart=D'2013.01.01 00:00'; // data copying st
//--- parameters for writing data to the file
input string          InpFileName="Volume.bin"; // file name
input string          InpDirectoryName="Data";  // directory name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   long     volume_buff[];
   datetime time_buff[];
   int      size;
//--- reset the error value
   ResetLastError();
//--- copy tick volumes for each bar
   if(CopyTickVolume(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finis
     {
      PrintFormat("Failed to copy values of the tick volume. Error code =
      return;
     }
//--- copy the time for each bar
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastErr
      return;
     }
//--- receive the buffer size
   size=ArraySize(volume_buff);
//--- open the file for writing the indicator values (if the file is absen
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- first, write the data sample size
      uint byteswritten=FileWriteLong(file_handle,(long)size);
      //--- check the number of bytes written
      if(byteswritten!=sizeof(long))
        {
         PrintFormat("Error in FileWriteLong. Error code=%d",GetLastError(
         //--- close the file
         FileClose(file_handle);
         return;
        }
      //--- write time and volume values to file
      for(int i=0;i<size;i++)
        {
         byteswritten=FileWriteLong(file_handle,(long)time_buff[i]);
         //--- check the number of bytes written
         if(byteswritten!=sizeof(long))
           {
            PrintFormat("Error in FileWriteLong. Error code=%d",GetLastErr
            //--- close the file
            FileClose(file_handle);
            return;
           }
         byteswritten=FileWriteLong(file_handle,volume_buff[i]);
         //--- check the number of bytes written
         if(byteswritten!=sizeof(long))
           {
            PrintFormat("Error in FileWriteLong. Error code=%d",GetLastErr
            //--- close the file
            FileClose(file_handle);
            return;
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
  }
```

See also

[Integer types](), [FileWriteInteger()]()

FileWriteString

The function writes the value of a string-type parameter into a BIN, CSV or TXT file starting from the current position of the file pointer. When writing to a CSV or TXT file: if there is a symbol in the string '\n' (LF) without previous character '\r' (CR), then before '\n' the missing '\r' is added.

```
uint  FileWriteString(    int            file_handle,     // File handle
   const string  text_string,     // string to write
   int           length=0       // number of symbols
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

text_string

[in] String.

length=0

[in] The number of characters that you want to write. This option is needed for writing a string into a BIN file. If the size is not specified, then the entire string without the trailer 0 is written. If you specify a size smaller than the length of the string, then a part of the string without the trailer 0 is written. If you specify a size greater than the length of the string, the string is filled by the appropriate number of zeros. For files of CSV and TXT type, this parameter is ignored and the string is written entirely.

Return Value

If successful the function returns the number of bytes written or 0 in case of error. To obtain information about the error call the GetLastError() function. The file pointer is shifted by the same number of bytes.

Note

Note that when writing to a file opened by the FILE_UNICODE flag (or without a flag FILE_ANSI), then the number of bytes written will be twice as large as the number of string characters written. When recording to a file opened with the FILE_ANSI flag, the number of bytes written will coincide with the number of string characters written.

Example:

```
//+------------------------------------------------------------------+
//|                                         Demo_FileWriteString.mq4 |
//|                        Copyright 2014, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
#property strict
//--- show the window of input parameters when launching the script
#property script_show_inputs
//--- parameters for receiving data from the terminal
input string             InpSymbolName="EURUSD";      // Currency pair
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;   // Time frame
input int                InpMAPeriod=14;              // MA period
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE; // Price type
//--- parameters for writing data to the file
input string             InpFileName="RSI.csv";       // File name
input string             InpDirectoryName="Data";     // Folder name
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   double   rsi_buff[];  // array of indicator values
   datetime date_buff[]; // array of the indicator dates
   int      rsi_size=0;  // size of the indicator arrays
//--- set indexing as timeseries
   ArraySetAsSeries(rsi_buff,true);
   ArraySetAsSeries(date_buff,true);
//--- reset last error code
   ResetLastError();
//--- copying the time from last 1000 bars
   int copied=CopyTime(NULL,0,0,1000,date_buff);
   if(copied<=0)
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastErr
      return;
     }
//--- prepare rsi_buff array
   ArrayResize(rsi_buff,copied);
//--- copy the values of RSI indicator
   for(int i=0;i<copied;i++)
     {
      rsi_buff[i]=iRSI(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpApplie
     }
//--- get size
   rsi_size=ArraySize(rsi_buff);
```

```
//--- open the file for writing the indicator values (if the file is abse
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is available for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_DAT
      //--- prepare additional variables
      string str="";
      bool   is_formed=false;
      //--- write dates of forming overbought and oversold areas
      for(int i=0;i<rsi_size;i++)
        {
         //--- check the indicator values
         if(rsi_buff[i]>=70 || rsi_buff[i]<=30)
           {
            //--- if the value is the first one in this area
            if(!is_formed)
              {
               //--- add the value and the date
               str=(string)rsi_buff[i]+"\t"+(string)date_buff[i];
               is_formed=true;
              }
            else
               str+="\t"+(string)rsi_buff[i]+"\t"+(string)date_buff[i];
            //--- move to the next loop iteration
            continue;
           }
         //--- check the flag
         if(is_formed)
           {
            //--- the string is formed, write it to the file
            FileWriteString(file_handle,str+"\r\n");
            is_formed=false;
           }
        }
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
  }
```

See also

String Type, StringFormat()

FileWriteStruct

The function writes into a bin-file contents of a structure passed as a parameter, starting from the current position of the file pointer.

```
uint  FileWriteStruct(    int            file_handle,        // File handle
   const void&   struct_object,       // link to an object
   int           size=-1              // size to be written in bytes
   );
```

Parameters

file_handle

[in] File descriptor returned by FileOpen().

struct_object

[in] Reference to the object of this structure. The structure should not contain strings, dynamic arrays or virtual functions.

size=-1

[in] Number of bytes that you want to record. If size is not specified or the specified number of bytes is greater than the size of the structure, the entire structure is written.

Return Value

If successful the function returns the number of bytes written or 0 in case of error. If successful, the number of bytes written corresponds to the size of the structure.To obtain information about the error call the GetLastError() function. The file pointer is shifted by the same number of bytes.

Example:

```
//+------------------------------------------------------------------+
//|                                         Demo_FileWriteStruct.mq4 |
//|                        Copyright 2014, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2014, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- show the window of input parameters when launching the script
#property script_show_inputs
#property strict
//--- parameters for receiving data from the terminal
input string          InpSymbolName="EURUSD";          // currency pair
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;       // time frame
input datetime        InpDateStart=D'2013.01.01 00:00'; // data copying st
//--- parameters for writing data to the file
input string          InpFileName="EURUSD.txt";          // file name
input string          InpDirectoryName="Data";          // directory name
//+------------------------------------------------------------------+
//| Structure for storing candlestick data                           |
//+------------------------------------------------------------------+
struct candlesticks
  {
   double            open;   // open price
   double            close;  // close price
   double            high;   // high price
   double            low;    // low price
   datetime          date;   // date
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime     date_finish=TimeCurrent();
   int          size;
   datetime     time_buff[];
   double       open_buff[];
   double       close_buff[];
   double       high_buff[];
   double       low_buff[];
   candlesticks cand_buff[];
//--- reset the error value
   ResetLastError();
//--- receive the time of the arrival of the bars from the range
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time
     {
      PrintFormat("Failed to copy time values. Error code = %d",GetLastErr
      return;
     }
//--- receive high prices of the bars from the range
   if(CopyHigh(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,high
     {
      PrintFormat("Failed to copy values of high prices. Error code = %d",
      return;
     }
//--- receive low prices of the bars from the range
   if(CopyLow(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,low_b
     {
      PrintFormat("Failed to copy values of low prices. Error code = %d",G
      return;
     }
//--- receive open prices of the bars from the range
   if(CopyOpen(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,open
     {
      PrintFormat("Failed to copy values of open prices. Error code = %d",
      return;
     }
//--- receive close prices of the bars from the range
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,clo
     {
      PrintFormat("Failed to copy values of close prices. Error code = %d"
      return;
     }
//--- define dimension of the arrays
   size=ArraySize(time_buff);
//--- save all data in the structure array
   ArrayResize(cand_buff,size);
   for(int i=0;i<size;i++)
     {
      cand_buff[i].open=open_buff[i];
      cand_buff[i].close=close_buff[i];
      cand_buff[i].high=high_buff[i];
      cand_buff[i].low=low_buff[i];
      cand_buff[i].date=time_buff[i];
     }
//--- open the file for writing the structure array to the file (if the fi
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("%s file is open for writing",InpFileName);
      PrintFormat("File path: %s\\Files\\",TerminalInfoString(TERMINAL_COM
      //--- prepare the counter of the number of bytes
      uint counter=0;
      //--- write array values in the loop
      for(int i=0;i<size;i++)
        {
         uint byteswritten=FileWriteStruct(file_handle,cand_buff[i]);
         //--- check the number of bytes written
         if(byteswritten!=sizeof(candlesticks))
           {
            PrintFormat("Error read data. Error code=%d",GetLastError());
            //--- close the file
            FileClose(file_handle);
            return;
           }
         else
```

```
            counter+=byteswritten;
         }
      PrintFormat("%d bytes of information is written to %s file",InpFileN
      PrintFormat("Total number of bytes: %d * %d * %d = %d, %s",size,5,8,
      //--- close the file
      FileClose(file_handle);
      PrintFormat("Data is written, %s file is closed",InpFileName);
     }
   else
      PrintFormat("Failed to open %s file, Error code = %d",InpFileName,Ge
  }
```

See also

[Structures and classes]()

FolderCreate

The function creates a folder in the Files directory (depending on the value of common_flag).

```
bool  FolderCreate(    string  folder_name,        // String with the name
   int     common_flag=0      // Scope
   );
```

Parameters

folder_name

[in] The name of the directory you want to create. Contains the full path to the folder.

common_flag=0

[in] Flag determining the location of the directory. If common_flag=FILE_COMMON, then the directory is in the shared folder for all client terminals \Terminal\Common\Files. Otherwise, the directory is in a local folder (MQL4\Files or MQL4\Tester\Files in case of testing).

Return Value

Returns true if successful, otherwise - false.

Note

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means, cannot be outside the file sandbox.

Example:

```
//+------------------------------------------------------------------+
//|                                         Demo_FolderCreate.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql15.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql15.com"
#property version   "1.00"
//--- Description
#property description "The script shows a sample use of FolderCreate()."
#property description "An external parameter defines the folder for creati
#property description "After running the script, a structure of folders is

//--- Show the dialog of input parameters when starting the script
```

```
#property script_show_inputs
//--- The input parameter defines the folder, in which the script is runni
input bool     common_folder=false; // A shared folder of all terminals
int            flag=0;              // The flag value determines the place
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string working_folder;
//--- Set the flag value, if the external parameter common_folder==true
   if(common_folder)
     {
      flag=FILE_COMMON;
      //--- Find the folder, in which we are working
      working_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH)+"\\MQL4\
     }
   else working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL4\\Fil
//--- The folder that will be created in the folder MQL4\Files
   string root="Folder_A";
   if(CreateFolder(working_folder,root,flag))
     {
      //--- Create a child folder in it Child_Folder_B1
      string folder_B1="Child_Folder_B1";
      string path=root+"\\"+folder_B1;          // Create a folder name ba
      if(CreateFolder(working_folder,path,flag))
        {
         //--- Create 3 more child folders in this folder
         string folder_C11="Child_Folder_C11";
         string child_path=path+"\\"+folder_C11;// Create a folder name ba
         CreateFolder(working_folder,child_path,flag);
         //--- The second child folder
         string folder_C12="Child_Folder_C12";
         child_path=path+"\\"+folder_C12;
         CreateFolder(working_folder,child_path,flag);

         //--- The third child folder
         string folder_C13="Child_Folder_C13";
         child_path=path+"\\"+folder_C13;
         CreateFolder(working_folder,child_path,flag);
        }
     }
//---
  }
//+------------------------------------------------------------------+
//| Tries to create a folder and shows a message                     |
//+------------------------------------------------------------------+
bool CreateFolder(string working_folder,string folder_path,int file_flag)
  {
//--- A debug message
   PrintFormat("folder_path=%s",folder_path);
//--- Trying to create a folder relative to path MQL4\Files
   if(FolderCreate(folder_path,file_flag))
     {
      //--- Show the entire path to the created folder
      PrintFormat("Folder %s has been created",working_folder+"\\"+folder_
      //--- Reset the error code
      ResetLastError();
      //--- Return successful result
      return true;
     }
   else
      PrintFormat("Failed to create folder %s. Error code %d",working_fold
//--- Failed
   return false;
  }
```

See also

FileOpen(), FolderClean(), FileCopy()

FolderDelete

The function removes the specified directory. If the folder is not empty, then it can't be removed.

```
bool  FolderDelete(    string  folder_name,        // String with the name
   int      common_flag=0      // Scope
   );
```

Parameters

folder_name

[in] The name of the directory you want to delete. Contains the full path to the folder.

common_flag=0

[in] Flag determining the location of the directory. If common_flag=FILE_COMMON, then the directory is in the shared folder for all client terminals \Terminal\Common\Files. Otherwise, the directory is in a local folder (MQL4\Files or MQL4\Tester\Files in the case of testing).

Return Value

Returns true if successful, otherwise false.

Note

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means, cannot be outside the file sandbox.

If the directory contains at least one file and/or subdirectory, then this directory can't be deleted, it must be cleared first. FolderClean() is used to clear a folder of all its files or subfolders.

Example:

```
//+------------------------------------------------------------------+
//|                                          Demo_FolderDelete.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- Description
#property description "The script shows a sample use of FolderDelete()."
```

```
#property description "First two folders are created; one of them is empty
#property description "When trying to delete a non-empty folder, an error

//--- Show the dialog of input parameters when starting the script
#property script_show_inputs
//--- Input parameters
input string   firstFolder="empty";    // An empty folder
input string   secondFolder="nonempty";// The folder, in which one file wi
string filename="delete_me.txt";       // The name of the file that will b
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- Write the file handle here
   int handle;
//--- Find out in what folder we are working
   string working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL4\\F
//--- A debug message
   PrintFormat("working_folder=%s",working_folder);
//--- Trying to create an empty folder relative to path MQL4\Files
   if(FolderCreate(firstFolder,0)) // 0 means that we are working in the l
     {
      //--- Enter the full path to the created folder
      PrintFormat("Folder %s has been created",working_folder+"\\"+firstFo
      //--- Reset the error code
      ResetLastError();
     }
   else
      PrintFormat("Failed to create folder %s. Error code %d",working_fold

//--- Now create a non-empty folder using the FileOpen() function
   string filepath=secondFolder+"\\"+filename;  // Form path to file that
   handle=FileOpen(filepath,FILE_WRITE|FILE_TXT); // Flag FILE_WRITE in th
   if(handle!=INVALID_HANDLE)
      PrintFormat("File %s has been opened for reading",working_folder+"\\
   else
      PrintFormat("Failed to create file %s in folder %s. Error code=",fil

   Comment(StringFormat("Prepare to delete folders %s and %s", firstFolder
//--- A small pause of 5 seconds to read a message in the chart
   Sleep(5000); // Sleep() cannot be used in indicators!

//--- Show a dialog and ask the user
   int choice=MessageBox(StringFormat("Do you want to delete folders %s an
                         "Deleting folders",
                         MB_YESNO|MB_ICONQUESTION); //  Two buttons - "Yes
```

```
//--- Run an action depending on the selected variant
   if(choice==IDYES)
     {
      //--- Delete the comment form the chart
      Comment("");
      //--- Add a message into the "Experts" journal
      PrintFormat("Trying to delete folders %s and %s",firstFolder, seconc
      ResetLastError();
      //--- Delete the empty folder
      if(FolderDelete(firstFolder))
         //--- The following message should appear since the folder is emp
         PrintFormat("Folder %s has been successfully deleted",firstFolder
      else
         PrintFormat("Failed to delete folder %s. Error code=%d", firstFol

      ResetLastError();
      //--- Delete the folder that contains a file
      if(FolderDelete(secondFolder))
         PrintFormat("Folder %s has been successfully deleted", secondFolc
      else
         //--- The following message should appear since the folder contai
         PrintFormat("Failed to delete folder %s. Error code=%d", secondFc
     }
   else
      Print("Deletion canceled");
//---
  }
```

See also

FileOpen(), FolderClean(), FileMove()

FolderClean

The function deletes all files in a specified folder.

```
bool  FolderClean(    string  folder_name,        // String with the name o
   int      common_flag=0      // Scope
   );
```

Parameters

folder_name

[in] The name of the directory where you want to delete all files. Contains the full path to the folder.

common_flag=0

[in] Flag determining the location of the directory. If common_flag = FILE_COMMON, then the directory is in the shared folder for all client terminals \Terminal\Common\Files. Otherwise, the directory is in a local folder(MQL4\files or MQL4\tester\files in case of testing).

Return Value

Returns true if successful, otherwise false.

Note

For security reasons, work with files is strictly controlled in the MQL4 language. Files with which file operations are conducted using MQL4 means, cannot be outside the file sandbox.

This function should be used with caution, since all the files and all subdirectories are deleted irretrievably.

Example:

```
//+------------------------------------------------------------------+
//|                                            Demo_FolderClean.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"
//--- Description
#property description "The script shows a sample use of FolderClean()."
#property description "First, files are created in the specified folder us
#property description "Then, before the files are deleted, a warning is sh
```

```
//--- Show the dialog of input parameters when starting the script
#property script_show_inputs
//--- Input parameters
input string   foldername="demo_folder";  // Create a folder in MQL4/Files
input int      files=5;                   // The number of files to create
//+-------------------------------------------------------------------+
//| Script program start function                                     |
//+-------------------------------------------------------------------+
void OnStart()
  {
   string name="testfile";
//--- First open or create files in the terminal data folder
   for(int N=0;N<files;N++)
     {
      //--- The name of the file in the foem of 'demo_folder\testfileN.txt
      string filemane=StringFormat("%s\\%s%d.txt",foldername,name,N);
      //--- Open a file with the flag for writing, in this case the 'demo_
      int handle=FileOpen(filemane,FILE_WRITE);
      //--- Find out if the FileOpen() function was successful
      if(handle==INVALID_HANDLE)
        {
         PrintFormat("Failed to create file %s. Error code",filemane,GetLa
         ResetLastError();
        }
      else
        {
         PrintFormat("File %s has been successfully opened",filemane);
         //--- The opened file is not needed any more, so close it
         FileClose(handle);
        }
     }

//--- Check the number of files in the folder
   int k=FilesInFolder(foldername+"\\*.*",0);
   PrintFormat("Totally the folder %s contains %d files",foldername,k);

//--- Show a dialog to ask the user
   int choice=MessageBox(StringFormat("You are going to delete %d files fr
                         "Deleting files from the folder",
                         MB_YESNO|MB_ICONQUESTION); //  Two buttons - "Yes
   ResetLastError();

//--- Run an action depending on the selected variant
   if(choice==IDYES)
     {
      //--- Start to delete files
      PrintFormat("Trying to delete all files from folder %s",foldername);
      if(FolderClean(foldername,0))
         PrintFormat("Files have been successfully deleted, %d files left
                     foldername,
                     FilesInFolder(foldername+"\\*.*",0));
      else
         PrintFormat("Failed to delete files from folder %s. Error code %d
     }
   else
      PrintFormat("Deletion canceled");
//---
  }
//+------------------------------------------------------------------+
//| Returns the number of files in the specified folder              |
//+------------------------------------------------------------------+
int FilesInFolder(string path,int flag)
  {
   int count=0;
   long handle;
   string filename;
//---
   handle=FileFindFirst(path,filename,flag);
//--- If at least one file found, search for more files
   if(handle!=INVALID_HANDLE)
     {
      //--- Show the name of the file
      PrintFormat("File %s found",filename);
      //--- Increase the counter of found files/folders
      count++;
      //--- Start search in all files/folders
      while(FileFindNext(handle,filename))
        {
         PrintFormat("File %s found",filename);
         count++;
        }
      //--- Do not forget to close the search handle upon completion
      FileFindClose(handle);
     }
   else // Failed to get the handle
     {
      PrintFormat("Files search in folder %s failed",path);
     }
//--- Return the result
   return count;
  }
```

See also

FileFindFirst(), FileFindNext(), FileFindClose()

FileOpenHistory

Opens file in the current history directory (terminal_directory\history\server_name) or in its subfolders.

```
int  FileOpenHistory(    int         filename,        // file name
   int         mode,             // open mode
   int         delimiter=';'    // delimiter
   );
```

Parameters

filename

[in] File name.

mode

[in] File open mode. Can be one or combination of values: FILE_BIN, FILE_CSV, FILE_READ, FILE_WRITE, FILE_SHARE_READ, FILE_SHARE_WRITE.

delimiter=';'

[in] Delimiter for csv files. By default, the ';' symbol will be passed.

Returned value

Returns the file handle for the opened file. If the function fails, the returned value is -1. To get the detailed error information, call the GetLastError() function.

Note

Client terminal can connect to servers of different brokerage companies. History data (HST files) for each brokerage company are stored in the corresponding subfolder of the terminal_directory\history folder. The function can be useful to form own history data for a non-standard symbol and/or period. The file formed in the history folder can be opened offline, not data pumping is needed to chart it.

In the new MQL4, FILE_SHARE_WRITE and FILE_SHARE_READ flags should explicitly be specified for shared use when opening files. If the flags are absent, the file is opened in exclusive mode and cannot be opened by anyone else till it is closed by the user who opened it (see "Offline Charts in the New MQL4").

Example:

```
   int handle=FileOpenHistory("USDX240.HST",FILE_BIN|FILE_WRITE|FILE_SHARE_
   if(handle<1)
     {
      Print("Cannot open file USDX240.HST");
      return(false);
     }
   // work with file
   // ...
   FileClose(handle);
```

Custom Indicators

This is the group functions used in the creation of custom indicators. These functions can't be used when writing Expert Advisors and Scripts.

Function	Action
HideTestIndicators	The function sets a flag hiding indicators called by the Expert Advisor
IndicatorSetDouble	Sets the value of an indicator property of the double type
IndicatorSetInteger	Sets the value of an indicator property of the int type
IndicatorSetString	Sets the value of an indicator property of the string type
SetIndexBuffer	Binds the specified indicator buffer with one-dimensional dynamic array of the double type
IndicatorBuffers	Allocates memory for buffers used for custom indicator calculations
IndicatorCounted	Returns the amount of bars not changed after the indicator had been launched last
IndicatorDigits	Sets precision format to visualize indicator values
IndicatorShortName	Sets the "short" name of a custom indicator to be shown in the DataWindow and in the chart subwindow
SetIndexArrow	Sets an arrow symbol for indicators line of the DRAW_ARROW type
SetIndexDrawBegin	Sets the bar number from which the drawing of the given indicator line must start
SetIndexEmptyValue	Sets drawing line empty value
SetIndexLabel	Sets drawing line description for showing in the DataWindow and in the tooltip
SetIndexShift	Sets offset for the drawing line
SetIndexStyle	Sets the new type, style, width and color for a given indicator line
SetLevelStyle	Sets a new style, width and color of horizontal levels of indicator to be output in a separate window
SetLevelValue	Sets a value for a given horizontal level of the indicator to be output in a separate window

Indicator properties can be set using the compiler directives or using functions. To better understand this, it is recommended that you study indicator styles in examples.

All the necessary calculations of a custom indicator must be placed in the predetermined function OnCalculate().

HideTestIndicators

The function sets a flag hiding indicators called by the Expert Advisor.

```
void  HideTestIndicators(    bool        hide      // flag
   );
```

Parameters

hide

[in] Hiding flag.

Returned value

None.

Note

After the Expert Advisor has been tested and the appropriate chart opened, the flagged indicators will not be drawn in the testing chart. Every indicator called will first be flagged with the current hiding flag. It must be noted that only those indicators can be drawn in the testing chart that are directly called from the expert under test.

Example:

```
   HideTestIndicators(true);
   MaCurrent=iMA(NULL,0,56,0,MODE_EMA,PRICE_CLOSE,0);
   MaPrevious=iMA(NULL,0,56,0,MODE_EMA,PRICE_CLOSE,1);
   HideTestIndicators(false);
```

IndicatorSetDouble

The function sets the value of the corresponding indicator property. Indicator property must be of the double type. There are two variants of the function.

Call with specifying the property identifier.

```
bool  IndicatorSetDouble(    int     prop_id,           // identifier
   double  prop_value         // value to be set
   );
```

Call with specifying the property identifier and modifier.

```
bool  IndicatorSetDouble(
   int     prop_id,           // identifier
   int     prop_modifier,     // modifier
   double  prop_value         // value to be set
   )
```

Parameters

prop_id

[in] Identifier of the indicator property. The value can be one of the values of the ENUM_CUSTOMIND_PROPERTY_DOUBLE enumeration.

prop_modifier

[in] Modifier of the specified property. Only level properties require a modifier. Numbering of levels starts from 0. It means that in order to set property for the second level you need to specify 1 (1 less than when using compiler directive).

prop_value

[in] Value of property.

Return Value

In case of successful execution, returns true, otherwise - false.

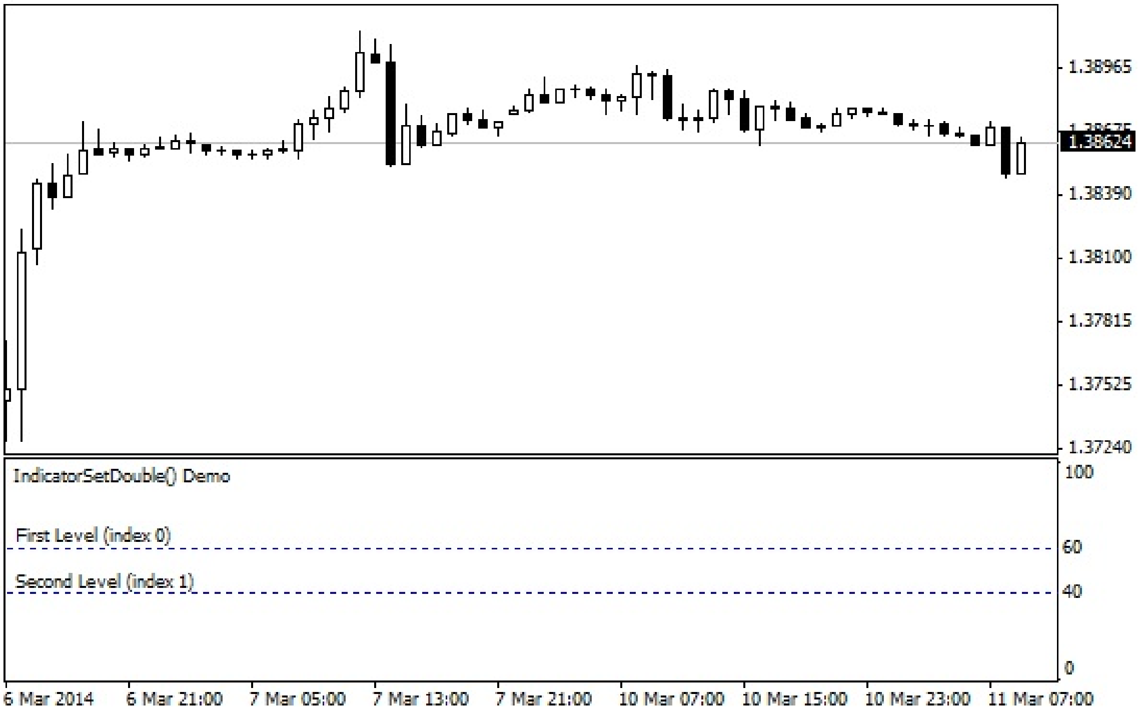
Note

Numbering of properties (modifiers) starts from 1 (one) when using the #property directive, while the function uses numbering from 0 (zero). In case the level number is set incorrectly, indicator display can differ from the intended one.

For example, the first level value for the indicator in a separate subwindow can be set in two ways:

- property indicator_level**1** 50 - the value of 1 is used for specifying the level number,
- IndicatorSetDouble(INDICATOR_LEVELVALUE, **0**, 50) - 0 is used for specifying the first level.

Example: indicator that turns upside down the values of levels on which the horizontal lines are placed.



```
#property indicator_separate_window
//--- set the maximum and minimum values for the indicator window
#property indicator_minimum  0
#property indicator_maximum  100
//--- display two horizontal levels in a separate indicator window
#property indicator_level1 25
#property indicator_level2 75
//--- set thickness of horizontal levels
#property indicator_levelwidth 1
//--- set style of horizontal levels
#property indicator_levelstyle STYLE_DOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- two levels
   IndicatorSetInteger(INDICATOR_LEVELS,2);
//--- set descriptions of horizontal levels
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
//--- set the short name for indicator
```

```
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetDouble() Demo");
//--- set clrBlue color for all levels
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,0,clrBlue);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int tick_counter=0;
   static double level1=25,level2=75;
   static int delta=1;
//--- calculate ticks
   tick_counter+=delta;
//--- check ranges
   if(tick_counter<0) delta=5;
   if(tick_counter>25) delta=-5;
//--- calculate new values
   level1+=delta;
   level2-=delta;
//--- set new values for levels
   IndicatorSetDouble(INDICATOR_LEVELVALUE,0,level1);
   IndicatorSetDouble(INDICATOR_LEVELVALUE,1,level2);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Drawing Styles

IndicatorSetInteger

The function sets the value of the corresponding indicator property. Indicator property must be of the int or color type. There are two variants of the function.

Call with specifying the property identifier.

```
bool  IndicatorSetInteger(    int   prop_id,           // identifier
   int   prop_value          // value to be set
   );
```

Call with specifying the property identifier and modifier.

```
bool  IndicatorSetInteger(
   int   prop_id,            // identifier
   int   prop_modifier,      // modifier
   int   prop_value          // value to be set
   )
```

Parameters

prop_id

[in] Identifier of the indicator property. The value can be one of the values of the ENUM_CUSTOMIND_PROPERTY_INTEGER enumeration.

prop_modifier

[in] Modifier of the specified property. Only level properties require a modifier.

prop_value

[in] Value of property.

Return Value

In case of successful execution, returns true, otherwise - false.

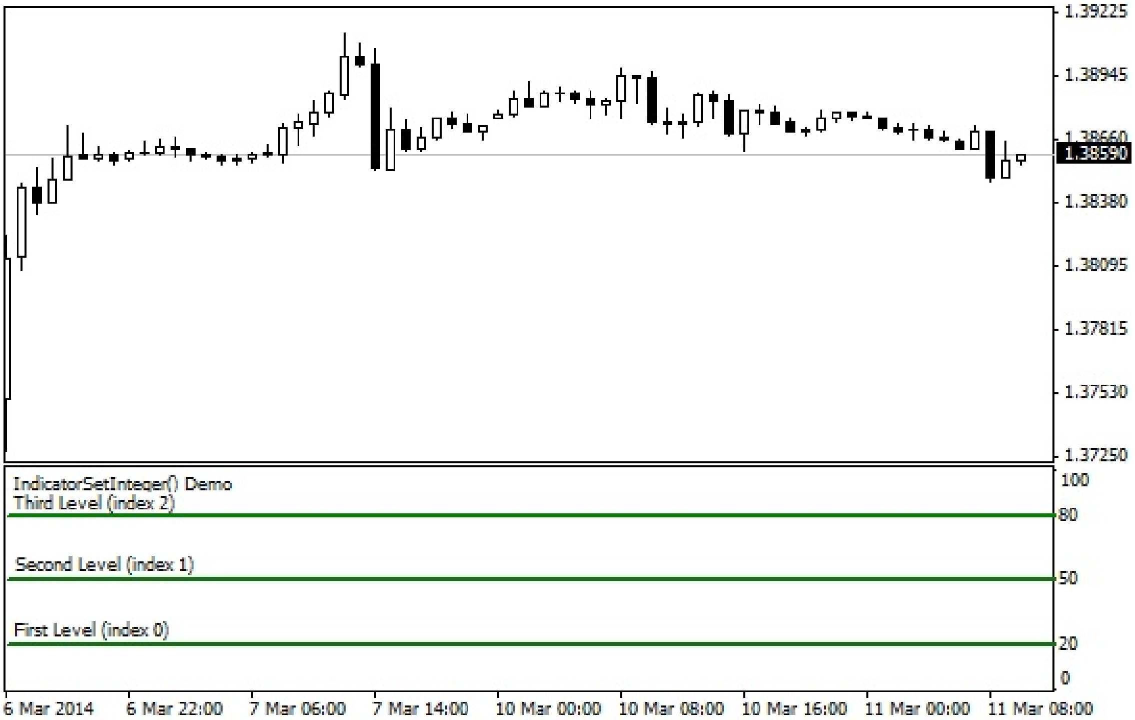
Note

Numbering of properties (modifiers) starts from 1 (one) when using the #property directive, while the function uses numbering from 0 (zero). In case the level number is set incorrectly, indicator display can differ from the intended one.

For example, in order to set thickness of the first horizontal line use zeroth index:

· IndicatorSetInteger(INDICATOR_LEVELWIDTH, **0**, 5) - index 0 is used to set thickness of the first level.

Example: indicator that turns upside down the values of levels on which the horizontal lines are placed.



```
#property indicator_separate_window
//--- set the maximum and minimum values for the indicator window
#property indicator_minimum 0
#property indicator_maximum 100
//--- display three horizontal levels in a separate indicator window
#property indicator_level1 20
#property indicator_level2 50
#property indicator_level3 80
//--- set thickness of horizontal levels
#property indicator_levelwidth 5
//--- set color of horizontal levels
#property indicator_levelcolor clrAliceBlue
//--- set style of horizontal levels
#property indicator_levelstyle STYLE_DOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- set descriptions of horizontal levels
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- set the short name for indicator
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetInteger() Demo");
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   static int tick_counter=0;
//--- calculate ticks
   tick_counter++;
//--- and calculate colors of horizontal levels depending on the tick cour
   ChangeLevelsColor(tick_counter,3,6,10); // three last parameters are sw
//--- modify style of horizontal levels
   ChangeLevelStyle(tick_counter);
//--- get width as the remainder of integer division of the ticks number b
   int width=tick_counter%5;
//--- iterate over all horizontal levels and set thickness
   IndicatorSetInteger(INDICATOR_LEVELWIDTH,0,width+1);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Set color of horizontal line in the separate indicator window    |
//+------------------------------------------------------------------+
void ChangeLevelsColor(int tick_number,// dividend, number to get the rema
                       int f_trigger,  // first divisor of color switching
                       int s_trigger,  // second divisor of color switchin
                       int t_trigger)  // third divisor of color switching
  {
   static color colors[3]={clrRed,clrBlue,clrGreen};
//--- index of color from the colors[] array
   int index=-1;
//--- calculate the number of color from the colors[] array to paint horiz
   if(tick_number%f_trigger==0)
      index=0;   // if tick_number divides by f_trigger without the remain
   if(tick_number%s_trigger==0)
      index=1;   // if tick_number divides by s_trigger without the remain
   if(tick_number%t_trigger==0)
```

```
      index=2;   // if tick_number divides by t_trigger without the remain
//--- if color is defined, set it
   if(index!=-1)
      IndicatorSetInteger(INDICATOR_LEVELCOLOR,0,colors[index]);
//---
  }
//+------------------------------------------------------------------+
//| Set style of horizontal line in the separate indicator window    |
//+------------------------------------------------------------------+
void ChangeLevelStyle(int tick_number) // number to get the remainder of d
  {
//--- array to store styles
   static ENUM_LINE_STYLE styles[5]=
     {STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT};
//--- index of style from the styles[] array
   int index=-1;
//--- calculate the number from the styles[] array to set style of horizon
   if(tick_number%50==0)
      index=5;   // if tick_number divides by 50 without the remainder, th
   if(tick_number%40==0)
      index=4;   // ... STYLE_DASHDOT
   if(tick_number%30==0)
      index=3;   // ... STYLE_DOT
   if(tick_number%20==0)
      index=2;   // ... STYLE_DASH
   if(tick_number%10==0)
      index=1;   // ... STYLE_SOLID
//--- if style is defined, set it
   if(index!=-1)
      IndicatorSetInteger(INDICATOR_LEVELSTYLE,0,styles[index]);
  }
```

See also

Custom Indicator Properties, Program Properties (#property), Drawing Styles

IndicatorSetString

The function sets the value of the corresponding indicator property. Indicator property must be of the string type. There are two variants of the function.

Call with specifying the property identifier.

```
bool  IndicatorSetString(    int     prop_id,          // identifier
   string  prop_value          // value to be set
   );
```

Call with specifying the property identifier and modifier.

```
bool  IndicatorSetString(
   int     prop_id,            // identifier
   int     prop_modifier,      // modifier
   string  prop_value          // value to be set
   )
```

Parameters

prop_id

[in] Identifier of the indicator property. The value can be one of the values of the ENUM_CUSTOMIND_PROPERTY_STRING enumeration.

prop_modifier

[in] Modifier of the specified property. Only level properties require a modifier.

prop_value

[in] Value of property.

Return Value

In case of successful execution, returns true, otherwise - false.

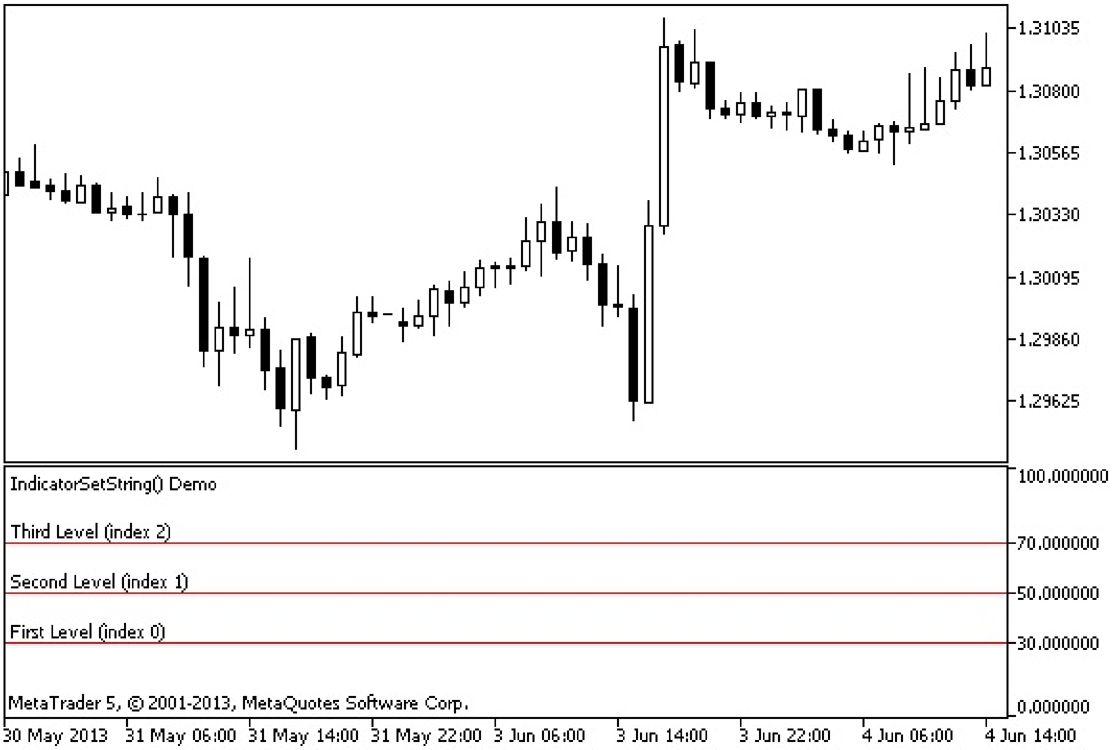
Note

Numbering of properties (modifiers) starts from 1 (one) when using the #property directive, while the function uses numbering from 0 (zero). In case the level number is set incorrectly, indicator display can differ from the intended one.

For example, in order to set description of the first horizontal line use zeroth index:

- IndicatorSetString(INDICATOR_LEVELTEXT, **0**, "First Level") - index 0 is used to set text description of the first level.

Example: indicator that sets text labels to the indicator horizontal lines.



```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- display three horizontal levels in a separate indicator window
#property indicator_level1 30
#property indicator_level2 50
#property indicator_level3 70
//--- set color of horizontal levels
#property indicator_levelcolor clrRed
//--- set style of horizontal levels
#property indicator_levelstyle STYLE_SOLID
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- set descriptions of horizontal levels
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- set the short name for indicator
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetString() Demo");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---

//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

See also

Custom Indicator Properties, Program Properties (#property)

SetIndexBuffer

The function binds a specified indicator buffer with one-dimensional dynamic array of the double type. There are two variants of the function.

```
bool  SetIndexBuffer(    int                    index,         // buffer i
   double                 buffer[],      // array
   ENUM_INDEXBUFFER_TYPE  data_type      // what will be stored
   );
```

Call without specifying of data type, stored in the indicator buffer.

```
bool  SetIndexBuffer(
   int                    index,         // buffer index
   double                 buffer[]       // array
   );
```

Parameters

index

[in] Number of the indicator buffer. The numbering starts with 0. The number must be less than the value declared in #property indicator_buffers.

buffer[]

[in] An array declared in the custom indicator program.

Return Value

If successful, returns true, otherwise - false.

Note

After binding, the dynamic array buffer[] will be indexed as in common arrays, even if the indexing of timeseries is pre-installed for the bound array. If you want to change the order of access to elements of the indicator array, use the ArraySetAsSeries() function **after binding the array using the SetIndexBuffer() function**. Please note that you can't change the size for dynamic arrays set as indicator buffers by the function SetIndexBuffer(). For indicator buffers, all operations of size changes are performed by the executing sub-system of the terminal.

Example:

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

See also

Custom Indicator Properties, Access to timeseries and indicators

IndicatorBuffers

Allocates memory for buffers used for custom indicator calculations.

```
bool  IndicatorBuffers(    int      count          // buffers
   );
```

Parameters

count

[in] Amount of buffers to be allocated. Should be within the range between indicator_buffers and 512 buffers.

Returned value

true, if the amount of buffers has been changed successfully, otherwise false.

Note

The amount of buffers cannot exceed 512 and be less than the value set in #property indicator_buffers. If a custom indicator requires additional buffers for counting, IndicatorBuffers() function should be used for specifying the total amount of buffers.

Example:

```
//+------------------------------------------------------------------+
//|                                                        Bulls.mq4 |
//|                   Copyright 2005-2013, MetaQuotes Software Corp. |
//|                                              https://www.mql4.com |
//+------------------------------------------------------------------+
#property copyright   "2005-2013, MetaQuotes Software Corp."
#property link        "https://www.mql4.com"
#property description "Bulls Power"
#property strict
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Silver
//--- input parameter
input int InpBullsPeriod=13;
//--- buffers
double ExtBullsBuffer[];
double ExtTempBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit(void)
  {
   string short_name;
//--- 1 additional buffer used for counting.
   IndicatorBuffers(2);
   IndicatorDigits(Digits);
//--- drawing style
   SetIndexStyle(0,DRAW_HISTOGRAM);
   SetIndexBuffer(0,ExtBullsBuffer);
   SetIndexBuffer(1,ExtTempBuffer);
//--- name for DataWindow and indicator subwindow label
   short_name="Bulls("+IntegerToString(InpBullsPeriod)+")";
   IndicatorShortName(short_name);
   SetIndexLabel(0,short_name);
  }
//+------------------------------------------------------------------+
//| Bulls Power                                                      |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   int limit=rates_total-prev_calculated;
//---
   if(rates_total<=InpBullsPeriod)
      return(0);
//---
   if(prev_calculated>0)
      limit++;
   for(int i=0; i<limit; i++)
     {
      ExtTempBuffer[i]=iMA(NULL,0,InpBullsPeriod,0,MODE_EMA,PRICE_CLOSE,i)
      ExtBullsBuffer[i]=high[i]-ExtTempBuffer[i];
     }
//---
   return(rates_total);
  }
```

See also

Custom Indicator Properties, Access to timeseries and indicators

IndicatorCounted

The function returns the amount of bars not changed after the indicator had been launched last.

```
int  IndicatorCounted();
```

Returned value

The amount of bars not changed after the indicator had been launched last.

Note

The most calculated bars do not need any recalculation. In most cases, same count of index values do not need for recalculation. The function is used to optimize calculating.

Example:

```
  int start()     {
     int limit;
     int counted_bars=IndicatorCounted();
  //---- check for possible errors
     if(counted_bars<0) return(-1);
  //---- the 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);
    }
```

See also

Custom Indicator Properties, Access to timeseries and indicators

IndicatorDigits

Sets precision format (the count of digits after decimal point) to visualize indicator values.

```
void  IndicatorDigits(    int       digits           // digits
   );
```

Parameters

digits

[in] Precision format, the count of digits after decimal point.

Returned value

None.

Note

The symbol price preicision is used by default, the indicator being attached to this symbol chart.

Example:

```
int init()
  {
//---- 2 additional buffers are used for counting.
   IndicatorBuffers(3);
//---- setting of drawing parameters
   SetIndexStyle(0,DRAW_HISTOGRAM,STYLE_SOLID,3);
   SetIndexDrawBegin(0,SignalSMA);
   IndicatorDigits(Digits+2);
//---- 3 allocated buffers of an indicator
   SetIndexBuffer(0,ind_buffer1);
   SetIndexBuffer(1,ind_buffer2);
   SetIndexBuffer(2,ind_buffer3);
//---- "short name" for DataWindow and indicator subwindow
   IndicatorShortName("OsMA("+FastEMA+","+SlowEMA+","+SignalSMA+")");
//---- initialization done
   return(0);
  }
```

See also

[Custom Indicator Properties]

IndicatorShortName

Sets the "short" name of a custom indicator to be shown in the DataWindow and in the chart subwindow.

```
void  IndicatorShortName(    string      name            // name
   );
```

Parameters

name

[in] New short name.

Returned value

None.

Example:

```
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);
  }
```

See also

Custom Indicator Properties

SetIndexArrow

Sets an arrow symbol for indicators line of the DRAW_ARROW type.

```
void  SetIndexArrow(    int       index,        // line index
   int       code           // code
   );
```

Parameters

index

[in] Line index. Must lie between 0 and 7.

code

[in] Symbol code from Wingdings font or predefined arrow constant.

Returned value

None.

Note

Arrow codes out of range 33 to 255 cannot be used.

Example:

```
int init()
  {
//---- 2 allocated indicator buffers
    SetIndexBuffer(0,ExtUppperBuffer);
    SetIndexBuffer(1,ExtLowerBuffer);
//---- drawing parameters setting
    SetIndexStyle(0,DRAW_ARROW);
    SetIndexArrow(0,217);
    SetIndexStyle(1,DRAW_ARROW);
    SetIndexArrow(1,218);
//---- displaying in the DataWindow
    SetIndexLabel(0,"Fractal Up");
    SetIndexLabel(1,"Fractal Down");
//---- initialization done
   return(0);
  }
```

See also

Custom Indicator Properties

SetIndexDrawBegin

Sets the bar number (from the data beginning) from which the drawing of the given indicator line must start.

```
void  SetIndexDrawBegin(    int       index,        // line index
   int       begin        // position
   );
```

Parameters

index

[in] Line index. Must lie between 0 and 7.

begin

[in] First drawing bar position number.

Returned value

None.

Note

The indicators are drawn from left to right. The indicator array values that are to the left of the given bar will not be shown in the chart or in the DataWindow. 0 will be set as default, and all data will be drawn.

Example:

```
//+------------------------------------------------------------------+
//|                                                    Alligator.mq4 |
//|                  Copyright 2005-2013, MetaQuotes Software Corp. |
//|                                              https://www.mql4.com |
//+------------------------------------------------------------------+
#property copyright   "2005-2013, MetaQuotes Software Corp."
#property link        "https://www.mql4.com"
#property description "Bill Williams' Alligator"
#property strict
//--- indicator settings
#property indicator_chart_window
#property indicator_buffers 3
#property indicator_color1  Blue
#property indicator_color2  Red
#property indicator_color3  Lime
//--- input parameters
input int InpJawsPeriod=13; // Jaws Period
input int InpJawsShift=8;   // Jaws Shift
input int InpTeethPeriod=8; // Teeth Period
input int InpTeethShift=5;  // Teeth Shift
input int InpLipsPeriod=5;  // Lips Period
input int InpLipsShift=3;   // Lips Shift
//--- indicator buffers
double ExtBlueBuffer[];
double ExtRedBuffer[];
double ExtLimeBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit(void)
  {
   IndicatorDigits(Digits);
//--- line shifts when drawing
   SetIndexShift(0,InpJawsShift);
   SetIndexShift(1,InpTeethShift);
   SetIndexShift(2,InpLipsShift);
//--- first positions skipped when drawing
   SetIndexDrawBegin(0,InpJawsShift+InpJawsPeriod);
   SetIndexDrawBegin(1,InpTeethShift+InpTeethPeriod);
   SetIndexDrawBegin(2,InpLipsShift+InpLipsPeriod);
//--- 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");
  }
//+------------------------------------------------------------------+
//| Bill Williams' Alligator                                         |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const datetime &time[],
                const double &open[],
                const double &high[],
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
   int limit=rates_total-prev_calculated;
//--- main loop
   for(int i=0; i<limit; i++)
     {
      //--- ma_shift set to 0 because SetIndexShift called abowe
      ExtBlueBuffer[i]=iMA(NULL,0,InpJawsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i
      ExtRedBuffer[i]=iMA(NULL,0,InpTeethPeriod,0,MODE_SMMA,PRICE_MEDIAN,i
      ExtLimeBuffer[i]=iMA(NULL,0,InpLipsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i
     }
//--- done
   return(rates_total);
  }
```

See also

[Custom Indicator Properties]

SetIndexEmptyValue

Sets drawing line empty value.

```
void  SetIndexEmptyValue(    int      index,        // line index
   double   value         // new "empty value"
   );
```

Parameters

index

[in] Line index. Must lie between 0 and 7.

value

[in] New "empty" value.

Returned value

None.

Note

Empty values are not drawn or shown in the DataWindow. By default, empty value is EMPTY_VALUE.

Example:

```
int init()
  {
//---- 2 allocated indicator buffers
    SetIndexBuffer(0,ExtUppperBuffer);
    SetIndexBuffer(1,ExtLowerBuffer);
//---- drawing parameters setting
    SetIndexStyle(0,DRAW_ARROW);
    SetIndexArrow(0,217);
    SetIndexStyle(1,DRAW_ARROW);
    SetIndexArrow(1,218);
//---- 0 value will not be displayed
    SetIndexEmptyValue(0,0.0);
    SetIndexEmptyValue(1,0.0);
//---- displaying in DataWindow
    SetIndexLabel(0,"Fractal Up");
    SetIndexLabel(1,"Fractal Down");
//---- initialization done
   return(0);
  }
```

See also

Custom Indicator Properties

SetIndexLabel

Sets drawing line description for showing in the DataWindow and in the tooltip.

```
void  SetIndexLabel(    int      index,        // line index
   string   text           // text
   );
```

Parameters

index

[in] Line index. Must lie between 0 and 7.

text

[in] Label text. NULL means that index value is not shown in the DataWindow.

Returned value

None.

Example:

```
//+--------------------------------------------------------------------+
//| 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,"Chikou Span");
//----
   return(0);
  }
```

See also

Custom Indicator Properties

SetIndexShift

Sets offset for the drawing line.

```
void  SetIndexShift(    int      index,        // line index
   int      shift          // shift
   );
```

Parameters

index

[in] Line index. Must lie between 0 and 7.

shift

[in] Shift value in bars.

Returned value

None.

Note

For positive values, the line drawing will be shifted to the right, otherwise it will be shifted to the left. I.e., the value calculated on the current bar will be drawn shifted relatively to the current bar.

Example:

```
//+-------------------------------------------------------------------+
//| 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);
  }
```

See also

[Custom Indicator Properties]

SetIndexStyle

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

```
void  SetIndexStyle(    int      index,        // line index
   int      type,          // line type
   int      style=EMPTY, // line style
   int      width=EMPTY, // line width
   color    clr=clrNONE  // line color
   );
```

Parameters

index

[in] Line index. Must lie between 0 and 7.

type

[in] Shape style. Can be one of Drawing shape styles listed.

style=EMPTY

[in] Drawing style. It is used for one-pixel thick lines. It can be one of the Drawing shape styles listed. EMPTY value means that the style will not be changed.

width=EMPTY

[in] Line width. Valid values are: 1,2,3,4,5. EMPTY value means that width will not be changed.

clr=clrNONE

[in] Line color. Absence of this parameter means that the color will not be changed.

Returned value

None.

Example:

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

See also

Custom Indicator Properties

SetLevelStyle

The function sets a new style, width and color of horizontal levels of indicator to be output in a separate window.

```
void  SetLevelStyle(    int      draw_style,        // drawing style
   int      line_width,        // line width
   color    clr                // color
   );
```

Parameters

draw_style

[in] Drawing style. Can be one of the Drawing shape styles listed. EMPTY value means that the style will not be changed.

line_width

[in] Line width. Valid values are 1,2,3,4,5. EMPTY value indicates that the width will not be changed.

clr

[in] Line color. Empty value CLR_NONE means that the color will not be changed.

Returned value

None.

Example:

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

See also

Custom Indicator Properties

SetLevelValue

The function sets a value for a given horizontal level of the indicator to be output in a separate window.

```
void  SetLevelValue(    int      level,        // level
   double   value         // value
   );
```

Parameters

level

[in] Level index (0-31).

value

[in] Value for the given indicator level.

Returned value

None.

Example:

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

See also

Custom Indicator Properties

Object Functions

This is the group of functions intended for working with graphic objects relating to any specified chart.

Function	Action
ObjectCreate	Creates an object of the specified type in a specified chart
ObjectName	Returns the name of an object by its index in the objects list
ObjectDelete	Removes the object having the specified name
ObjectsDeleteAll	Removes all objects of the specified type from the specified chart subwindow
ObjectFind	Searches for an object having the specified name
ObjectGetTimeByValue	Returns the time value for the specified object price value
ObjectGetValueByTime	Returns the price value of an object for the specified time
ObjectMove	Changes the coordinates of the specified object anchor point
ObjectsTotal	Returns the number of objects of the specified type
ObjectGetDouble	Returns the double value of the corresponding object property
ObjectGetInteger	Returns the integer value of the corresponding object property
ObjectGetString	Returns the string value of the corresponding object property
ObjectSetDouble	Sets the value of the corresponding object property
ObjectSetInteger	Sets the value of the corresponding object property
ObjectSetString	Sets the value of the corresponding object property
TextSetFont	Sets the font for displaying the text using drawing methods (Arial 20 used by default)
TextOut	Transfers the text to the custom array (buffer) designed for creation of a graphical resource
TextGetSize	Returns the string's width and height at the current font settings
ObjectDescription	Returns the object description

ObjectGet	Returns the value of the specified object property
ObjectGetFiboDescription	Returns the level description of a Fibonacci object
ObjectGetShiftByValue	Calculates and returns bar index for the given price
ObjectGetValueByShift	Calculates and returns the price value for the specified bar
ObjectSet	Changes the value of the specified object property
ObjectSetFiboDescription	Sets a new description to a level of a Fibonacci object
ObjectSetText	Changes the object description
ObjectType	Returns the object type

Every graphical object should have a name unique within one chart, including its subwindows. Changing of a name of a graphic object generates two events: event of deletion of an object with the old name, and event of creation of an object with a new name.

After an object is created or an object property is modified it is recommended to call the ChartRedraw() function, which commands the client terminal to forcibly draw a chart (and all visible objects in it).

ObjectCreate

The function creates an object with the specified name, type, and the initial coordinates in the specified chart subwindow of the specified chart. There are two variants of the function:

```
bool  ObjectCreate(    long             chart_id,       // chart ID
   string          object_name,   // object name
   ENUM_OBJECT     object_type,   // object type
   int             sub_window,    // window index
   datetime        time1,         // time of the first anchor point
   double          price1,        // price of the first anchor point
   ...
   datetime        timeN=0,       // time of the N-th anchor point
   double          priceN=0       // price of the N-th anchor point
   );
```

The function creates an object with the specified name, type, and the initial coordinates in the specified chart subwindow:

```
bool  ObjectCreate(
   string          object_name,   // object name
   ENUM_OBJECT     object_type,   // object type
   int             sub_window,    // window index
   datetime        time1,         // time of the first anchor point
   double          price1,        // price of the first anchor point
   datetime        time2=0,       // time of the second anchor point
   double          price2=0,      // price of the second anchor point
   datetime        time3=0,       // time of the third anchor point
   double          price3=0       // price of the third anchor point
   );
```

Parameters

chart_id

[in] Chart identifier.

object_name

[in] Name of the object. The name must be unique within a chart, including its subwindows.

object_type

[in] Object type. The value can be one of the values of the ENUM_OBJECT enumeration.

sub_window

[in] Number of the chart subwindow. 0 means the main chart window. The specified subwindow must exist (window index must be greater or equal to 0 and less than WindowsTotal()), otherwise the function returns false.

time1

[in] The time coordinate of the first anchor point.

price1

[in] The price coordinate of the first anchor point.

time2=0

[in] The time coordinate of the second anchor point.

price2=0

[in] The price coordinate of the second anchor point.

time3=0

[in] The time coordinate of the third anchor point.

price3=0

[in] The price coordinate of the third anchor point.

timeN=0

[in] The time coordinate of the N-th anchor point.

priceN=0

[in] The price coordinate of the N-th anchor point.

Return Value

Returns true or false depending on whether the object is created or not. To read more about the error call GetLastError(). If the object has been created already, the function tries to change its coordinates.

Note

An object name should not exceed 63 characters. Characters not belonging to the current code page are not allowed (characters that cannot be converted from Unicode to ANSI are replaced with '?'). If programs are to be distributed among users with different code pages, we strongly recommend using Latin characters in object names.

Objects of the OBJ_LABEL type ignore the coordinates. Use the ObjectSet() function to set up the OBJPROP_XDISTANCE and OBJPROP_YDISTANCE properties. The chart sub-windows (if there are sub-windows with indicators in the chart) are numbered starting from 1. The chart main window always exists and has the 0 index. Coordinates must be passed in pairs: time and price. For example, the OBJ_VLINE object needs only time, but price (any value) must be passed, as well.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int start()
  {
   int i;
   string obj_name="label_object";
   long current_chart_id=ChartID();
//--- creating label object (it does not have time/price coordinates)
   if(!ObjectCreate(current_chart_id,obj_name,OBJ_LABEL,0,0,0))
     {
      Print("Error: can't create label! code #",GetLastError());
      return(0);
     }
//--- set color to Red
   ObjectSetInteger(current_chart_id,obj_name,OBJPROP_COLOR,clrRed);
//--- move object down and change its text
   for(i=0; i<200; i++)
     {
      //--- set text property
      ObjectSetString(current_chart_id,obj_name,OBJPROP_TEXT,StringFormat(
      //--- set distance property
      ObjectSet(obj_name,OBJPROP_YDISTANCE,i);
      //--- forced chart redraw
      ChartRedraw(current_chart_id);
      Sleep(10);
     }
//--- set color to Blue
   ObjectSetInteger(current_chart_id,obj_name,OBJPROP_COLOR,clrBlue);
//--- move object up and change its text
   for(i=200; i>0; i--)
     {
      //--- set text property
      ObjectSetString(current_chart_id,obj_name,OBJPROP_TEXT,StringFormat(
      //--- set distance property
      ObjectSet(obj_name,OBJPROP_YDISTANCE,i);
      //--- forced chart redraw
      ChartRedraw(current_chart_id);
      Sleep(10);
     }
//--- delete object
   ObjectDelete(obj_name);
   return(0);
  }
```

See also

Object types

ObjectName

The function returns the name of the corresponding object by its index in the objects list.

```
string  ObjectName(    int    object_index    // object index
   );
```

Parameters

object_index

[in] Object index. This value must be greater or equal to 0 and less than ObjectsTotal().

Return Value

Name of the object is returned in case of success. To get the detailed error information, one has to call the GetLastError() function.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   int i;
   long current_chart_id=ChartID();
//--- creates several objects of label type
   for(i=0; i<300; i+=10)
     {
      string obj_name="label_object"+IntegerToString(i);
      //--- creating label object (it does not have time/price coordinates
      if(ObjectCreate(obj_name,OBJ_LABEL,0,0,0))
        {
         PrintFormat("Object %s created.",obj_name);
         //--- set random color
         ObjectSetInteger(current_chart_id,obj_name,OBJPROP_COLOR,MathRand
         //--- set text property
         ObjectSetString(current_chart_id,obj_name,OBJPROP_TEXT,StringForm
         //--- set distance property
         ObjectSet(obj_name,OBJPROP_XDISTANCE,i);
         ObjectSet(obj_name,OBJPROP_YDISTANCE,i);
         //--- forced chart redraw
         ChartRedraw(current_chart_id);
         Sleep(10);
        }
      else
        {
         Print("Error: can't create label! code #",GetLastError());
        }
     }
//--- sleep to see the objects created
   Sleep(3000);
//--- show all objects
   int obj_total=ObjectsTotal();
   PrintFormat("Total %d objects",obj_total);
   string name;
   for(i=0;i<obj_total;i++)
     {
      name=ObjectName(i);
      PrintFormat("%d object: Object name - %s",i,name);
     }
//--- delete all objects
   ObjectsDeleteAll();
  }
```

ObjectDelete

The function removes the object with the specified name at the specified chart. There are two variants of the function:

```
bool  ObjectDelete(    long      chart_id,      // chart ID
   string   object_name    // object name
   );
```

The function removes the object with the specified name:

```
bool  ObjectDelete(
   string   object_name    // object name
   );
```

Parameters

chart_id

[in] Chart identifier.

object_name

[in] Name of object to be deleted.

Return Value

Returns true if the removal was successful, otherwise returns false. To read more about the error call GetLastError().

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int start()
  {
   int i;
   long current_chart_id=ChartID();
//--- creates several objects of label type
   for(i=0; i<300; i+=10)
     {
      string obj_name="label_object"+IntegerToString(i);
      //--- creating label object (it does not have time/price coordinates
      if(ObjectCreate(obj_name,OBJ_LABEL,0,0,0))
        {
         PrintFormat("Object %s created.",obj_name);
         //--- set random color
         ObjectSetInteger(current_chart_id,obj_name,OBJPROP_COLOR,MathRand
         //--- set text property
         ObjectSetString(current_chart_id,obj_name,OBJPROP_TEXT,StringForm
         //--- set distance property
         ObjectSet(obj_name,OBJPROP_XDISTANCE,i);
         ObjectSet(obj_name,OBJPROP_YDISTANCE,i);
         //-- forced chart redraw
         ChartRedraw(current_chart_id);
         Sleep(10);
        }
      else
        {
         Print("Error: can't create label! code #",GetLastError());
         return(0);
        }
     }
//--- sleep to see the objects created
   Sleep(3000);
//--- delete all objects
   int obj_total=ObjectsTotal();
   PrintFormat("Total %d objects",obj_total);
   for(i=obj_total-1;i>=0;i--)
     {
      string name=ObjectName(i);
      PrintFormat("object %d: %s",i,name);
      ObjectDelete(name);
     }
   return(0);
  }
```

ObjectsDeleteAll

Removes all objects from the specified chart, specified chart subwindow, of the specified type.

```
int  ObjectsDeleteAll(    long    chart_id,           // chart ID
   int     sub_window=EMPTY,   // window index
   int     object_type=EMPTY   // object type
   );
```

Removes all objects of the specified type from the specified chart subwindow.

```
int  ObjectsDeleteAll(
   int     sub_window=EMPTY,   // window index
   int     object_type=EMPTY   // object type
   );
```

Removes all objects of the specified type using prefix in object names.

```
int  ObjectsDeleteAll(
   long           chart_id,   // chart ID
   const string     prefix,   // prefix in object name
   int     sub_window=EMPTY,   // window index
   int     object_type=EMPTY   // object type
   );
```

Parameters

chart_id

[in] Chart identifier.

prefix

[in] Prefix in object names. All objects whose names start with this set of characters will be removed from chart. You can specify prefix as 'name' or 'name*' both variants will work the same. If an empty string is specified as the prefix, objects with all possible names will be removed.

sub_window=EMPTY

[in] Number of the chart window. Must be greater or equal to -1 (-1 mean all subwindows, 0 means the main chart window) and less than WindowsTotal().

object_type=EMPTY

[in] Type of the object. The value can be one of the values of the ENUM_OBJECT enumeration. EMPTY (-1) means all types.

Return Value

Returns the number of deleted objects. To read more about the error call GetLastError().

Example:

```
  ObjectsDeleteAll(2, OBJ_HLINE); // delete all horizontal lines from the
  ObjectsDeleteAll(2);            // delete all objects from the 2nd subwi
  ObjectsDeleteAll();             // delete all objects from chart.
```

ObjectFind

The function searches for an object having the specified name. There are two variants of the function:

```
int  ObjectFind(    long      chart_id,      // chart ID
   string   object_name   // object name
   );
```

The function searches the object with the specified name:

```
int  ObjectFind(
   string   object_name   // object name
   );
```

Parameters

chart_id

[in] Chart identifier.

object_name

[in] The name of the object to find.

Return Value

If successful the function returns the number of the subwindow (0 means the main window of the chart), in which the object is found. If the object is not found, the function returns a negative number. To read more about the error call GetLastError().

Note

The chart sub-windows (if there are sub-windows with indicators in the chart) are numbered starting from 1. The chart main window always exists and has the 0 index.

Example:

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

ObjectGetTimeByValue

The function returns the time value for the specified price value of the specified object.

```
datetime  ObjectGetTimeByValue(   long     chart_id,     // chart ID
   string   object_name,   // object name
   double   value,         // price
   int      line_id=0      // line identifier
   );
```

Parameters

chart_id

[in] Chart identifier.

object_name

[in] Name of the object.

value

[in] Price value.

line_id=0

[in] Line identifier.

Return Value

The time value for the specified price value of the specified object.

Note

An object can have several values in one price coordinate, therefore it is necessary to specify the line number. This function applies only to the following objects:

- Trendline (OBJ_TREND)
- Trendline by angle (OBJ_TRENDBYANGLE)
- Gann line (OBJ_GANNLINE)
- Equidistant channel (OBJ_CHANNEL) - 2 lines
- Linear regression channel (OBJ_REGRESSION) - 3 lines
- Standard deviation channel (OBJ_STDDEVCHANNEL) - 3 lines

See also

Object Types

ObjectGetValueByTime

The function returns the price value for the specified time value of the specified object.

```
double  ObjectGetValueByTime(   long       chart_id,      // chart ID
   string     object_name,   // object name
   datetime   time,          // time
   int        line_id=0      // line ID
   );
```

Parameters

chart_id

[in] Chart identifier.

object_name

[in] Name of the object.

time

[in] Time value.

line_id=0

[in] Line identifier.

Return Value

The price value for the specified time value of the specified object.

Note

An object can have several values in one price coordinate, therefore it is necessary to specify the line number. This function applies only to the following objects:

- Trendline (OBJ_TREND)
- Trendline by angle (OBJ_TRENDBYANGLE)
- Gann line (OBJ_GANNLINE)
- Equidistant channel (OBJ_CHANNEL) - 2 lines
- Linear regression channel (OBJ_REGRESSION) - 3 lines
- Standard deviation channel (OBJ_STDDEVCHANNEL) - 3 lines

See also

Object Types

ObjectMove

The function changes coordinates of the specified anchor point of the object at the specified chart. There are two variants of the function:

```
bool  ObjectMove(    string      object_name,    // object name
   int         point_index,    // anchor point number
   datetime    time,           // Time
   double      price           // Price
   );
```

The function changes coordinates of the specified anchor point of the object.

```
bool  ObjectMove(
   string      object_name,    // object name
   int         point_index,    // anchor point number
   datetime    time,           // Time
   double      price           // Price
   );
```

Parameters

object_name

[in] Name of the object.

point_index

[in] Index of the anchor point. The number of anchor points depends on the type of object.

time

[in] Time coordinate of the selected anchor point.

price

[in] Price coordinate of the selected anchor point.

Return Value

If successful, returns true, in case of failure returns false. To read more about the error call GetLastError().

Note

The function moves an object coordinate in the chart. Objects can have from one to three coordinates depending on their types. The object coordinates are numbered starting from 0.

Example:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
int start()
  {
   string obj_name="trend_line";
   long current_chart_id=ChartID();
//---
   datetime t1=Time[0];
   double p1=Close[0];
//---
   datetime t2=Time[1];
   double p2=Close[1];
//--- creating trend line object
   if(!ObjectCreate(obj_name,OBJ_TREND,0,t1,p1,t2,p2))
     {
      Print("Error: can't create trend line! code #",GetLastError());
      return(0);
     }
//--- set color to Red
   ObjectSetInteger(current_chart_id,obj_name,OBJPROP_COLOR,clrRed);
//--- moving of the trend line
   for(int i=1; i<200; i++)
     {
      t2=Time[i];
      p2=Close[i];
      //--- move the 2nd anchor point of the trend line
      ObjectMove(obj_name,1,t2,p2);
      //--- forced chart redraw
      ChartRedraw(current_chart_id);
      Sleep(100);
     }
//--- sleep to see the object
   Sleep(3000);
//--- delete object
   ObjectDelete(obj_name);
   return(0);
  }
```

ObjectsTotal

The function returns the number of objects of the specified type in the specified chart. There are two variants of the function:

```
int  ObjectsTotal(   long  chart_id,          // chart identifier
   int    sub_window=-1,      // window index
   int    type=-1             // object type
   );
```

The function returns the number of objects of the specified type:

```
int  ObjectsTotal(
   int     type=EMPTY         // object type
   );
```

Parameters

chart_id

[in] Chart identifier.

sub_window=-1

[in] Number of the chart subwindow. 0 means the main chart window, -1 means all the subwindows of the chart, including the main window.

type=-1

[in] Type of the object. The value can be one of the values of the ENUM_OBJECT enumeration. EMPTY(-1) means all types.

Return Value

The number of objects.

Example:

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

ObjectGetDouble

The function returns the value of the corresponding object property. The object property must be of the double type. There are 2 variants of the function.

1. Immediately returns the property value.

```
double  ObjectGetDouble(    long      chart_id,          // chart identifier
   string   object_name,      // object name
   int      prop_id,          // property identifier
   int      prop_modifier=0   // property modifier, if required
   );
```

2. Returns true or false, depending on the success of the function. If successful, the property value is placed to a receiving variable passed by reference by the last parameter.

```
bool  ObjectGetDouble(
   long      chart_id,        // chart identifier
   string    object_name,     // object name
   int       prop_id,         // property identifier
   int       prop_modifier,   // property modifier
   double&   double_var       // here we accept the property value
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

object_name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_DOUBLE enumeration.

prop_modifier

[in] Modifier of the specified property. For the first variant, the default modifier value is equal to 0. Most properties do not require a modifier. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

double_var

[out] Variable of the double type that received the value of the requested

property.

Return Value

Value of the double type for the first calling variant.

For the second variant the function returns true, if this property is maintained and the value has been placed into the double_var variable, otherwise returns false. To read more about the error call GetLastError().

ObjectGetInteger

The function returns the value of the corresponding object property. The object property must be of the datetime, int, color, bool or char type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long  ObjectGetInteger(    long      chart_id,          // chart identifier
   string   object_name,       // object name
   int      prop_id,           // property identifier
   int      prop_modifier=0    // property modifier, if required
   );
```

2. Returns true or false, depending on the success of the function. If successful, the property value is placed to a receiving variable passed by reference by the last parameter.

```
bool  ObjectGetInteger(
   long     chart_id,          // chart identifier
   string   object_name,       // object name
   int      prop_id,           // property identifier
   int      prop_modifier,     // property modifier
   long&    long_var           // here we accept the property value
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

object_name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_INTEGER enumeration.

prop_modifier

[in] Modifier of the specified property. For the first variant, the default modifier value is equal to 0. Most properties do not require a modifier. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

long_var

[out] Variable of the long type that receives the value of the requested

property.

Return Value

The long value for the first calling variant.

For the second variant the function returns true, if this property is maintained and the value has been placed into the long_var variable, otherwise returns false. To read more about the error call GetLastError().

ObjectGetString

The function returns the value of the corresponding object property. The object property must be of the string type. There are 2 variants of the function.

1. Immediately returns the property value.

```
string  ObjectGetString(    long    chart_id,          // chart identifier
   string   object_name,       // object name
   int      prop_id,           // property identifier
   int      prop_modifier=0    // property modifier, if required
   );
```

2. Returns true or false, depending on the success of the function. If successful, the property value is placed to a receiving variable passed by reference by the last parameter.

```
bool  ObjectGetString(
   long     chart_id,          // chart identifier
   string   object_name,       // object name
   int      prop_id,           // property identifier
   int      prop_modifier,     // property modifier
   string&  string_var         // here we accept the property value
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

object_name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_STRING enumeration.

prop_modifier

[in] Modifier of the specified property. For the first variant, the default modifier value is equal to 0. Most properties do not require a modifier. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

string_var

[out] Variable of the string type that receives the value of the requested

properties.

Return Value

String value for the first version of the call.

For the second version of the call returns true, if this property is maintained and the value has been placed into the string_var variable, otherwise returns false. To read more about the error call GetLastError().

ObjectSetDouble

The function sets the value of the corresponding object property. The object property must be of the double type. There are 2 variants of the function.

Setting property value, without modifier.

```
bool  ObjectSetDouble(    long     chart_id,        // chart identifier
   string   object_name,     // object name
   int      prop_id,         // property
   double   prop_value       // value
   );
```

Setting a property value indicating the modifier.

```
bool  ObjectSetDouble(
   long     chart_id,        // chart identifier
   string   object_name,     // object name
   int      prop_id,         // property
   int      prop_modifier,   // modifier
   double   prop_value       // value
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

object_name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_DOUBLE enumeration.

prop_modifier

[in] Modifier of the specified property. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

prop_value

[in] The value of the property.

Return Value

The function returns true only if the command to change properties of a graphical object has been sent to a chart successfully. Otherwise it returns false. To read more about the error call GetLastError().

Example of creating a Fibonacci object and adding a new level in it

```
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- auxiliary arrays
   double high[],low[],price1,price2;
   datetime time[],time1,time2;
//--- Copy the open prices - 100 latest bars are enough
   int copied=CopyHigh(Symbol(),0,0,100,high);
   if(copied<=0)
     {
      Print("Failed to copy the values of the High price series");
      return;
     }
//--- Copy the close price - 100 latest bars are enough
   copied=CopyLow(Symbol(),0,0,100,low);
   if(copied<=0)
     {
      Print("Failed to copy the values of the Low price series");
      return;
     }
//--- Copy the open time for the last 100 bars
   copied=CopyTime(Symbol(),0,0,100,time);
   if(copied<=0)
     {
      Print("Failed to copy the values of the price series of Time");
      return;
     }
//--- Organize access to the copied data as to timeseries - backwards
   ArraySetAsSeries(high,true);
   ArraySetAsSeries(low,true);
   ArraySetAsSeries(time,true);

//--- Coordinates of the first anchor point of the Fibo object
   price1=high[70];
   time1=time[70];
//--- Coordinates of the second anchor point of the Fibo object
   price2=low[50];
   time2=time[50];

//--- Time to create the Fibo object
   bool created=ObjectCreate(0,"Fibo",OBJ_FIBO,0,time1,price1,time2,price2
   if(created) // If the object is created successfully
     {
```

```
    //--- set the color of Fibo levels
    ObjectSetInteger(0,"Fibo",OBJPROP_LEVELCOLOR,Blue);
    //--- by the way, how much Fibo levels do we have?
    int levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
    Print("Fibo levels before = ",levels);
    //---output to the Journal => number of level:value level_desription
    for(int i=0;i<levels;i++)
      {
       Print(i,": ",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
             "  ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
      }
    //--- Try to increase the number of levels per unit
    bool modified=ObjectSetInteger(0,"Fibo",OBJPROP_LEVELS,levels+1);
    if(!modified) // failed to change the number of levels
      {
       Print("Failed to change the number of levels of Fibo, error ",Get
      }
    //--- just inform
    Print("Fibo levels after = ",ObjectGetInteger(0,"Fibo",OBJPROP_LEVEL
    //--- set a value for a newly created level
    bool added=ObjectSetDouble(0,"Fibo",OBJPROP_LEVELVALUE,levels,133);
    if(added) // managed to set a value for the level
      {
       Print("Successfully set one more Fibo level");
       //--- Also do not forget to set the level description
       ObjectSetString(0,"Fibo",OBJPROP_LEVELTEXT,levels,"my level");
       ChartRedraw(0);
       //--- Get the actual value of the number of levels in the Fibo ob
       levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
       Print("Fibo levels after adding = ",levels);
       //--- once again output all levels - just to make sure
       for(int i=0;i<levels;i++)
         {
          Print(i,":",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
                "  ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
         }
      }
    else // Fails if you try to increase the number of levels in the Fib
      {
       Print("Failed to set one more Fibo level. Error ",GetLastError())
      }
   }
}
```

See also

Object Types, Object Properties

ObjectSetInteger

The function sets the value of the corresponding object property. The object property must be of the datetime, int, color, bool or char type. There are 2 variants of the function.

Setting property value, without modifier:

```
bool  ObjectSetInteger(    long     chart_id,       // chart identifier
   string   object_name,    // object name
   int      prop_id,        // property
   long     prop_value      // value
   );
```

Setting a property value indicating the modifier:

```
bool  ObjectSetInteger(
   long     chart_id,        // chart identifier
   string   object_name,     // object name
   int      prop_id,         // property
   int      prop_modifier,   // modifier
   long     prop_value       // value
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

object_name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_INTEGER enumeration.

prop_modifier

[in] Modifier of the specified property. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

prop_value

[in] The value of the property.

Return Value

The function returns true only if the command to change properties of a graphical object has been sent to a chart successfully. Otherwise it returns false. To read more about the error call GetLastError().

An example of how to create a table of Web colors

```
//+------------------------------------------------------------------+
//|                                               Table of Web Colors|
//|                         Copyright 2011, MetaQuotes Software Corp |
//|                                        http://www.metaquotes.net |
//+------------------------------------------------------------------+
#define X_SIZE 140      // width of an edit object
#define Y_SIZE 33       // height of an edit object
//+------------------------------------------------------------------+
//| Array of web colors                                              |
//+------------------------------------------------------------------+
color ExtClr[140]=
  {
   clrAliceBlue,clrAntiqueWhite,clrAqua,clrAquamarine,clrAzure,clrBeige,cl
   clrBlue,clrBlueViolet,clrBrown,clrBurlyWood,clrCadetBlue,clrChartreuse,
   clrCornsilk,clrCrimson,clrCyan,clrDarkBlue,clrDarkCyan,clrDarkGoldenrod
   clrDarkMagenta,clrDarkOliveGreen,clrDarkOrange,clrDarkOrchid,clrDarkRed
   clrDarkSlateBlue,clrDarkSlateGray,clrDarkTurquoise,clrDarkViolet,clrDee
   clrDodgerBlue,clrFireBrick,clrFloralWhite,clrForestGreen,clrFuchsia,clr
   clrGoldenrod,clrGray,clrGreen,clrGreenYellow,clrHoneydew,clrHotPink,clr
   clrLavender,clrLavenderBlush,clrLawnGreen,clrLemonChiffon,clrLightBlue,
   clrLightGoldenrod,clrLightGreen,clrLightGray,clrLightPink,clrLightSalmo
   clrLightSlateGray,clrLightSteelBlue,clrLightYellow,clrLime,clrLimeGreen
   clrMediumAquamarine,clrMediumBlue,clrMediumOrchid,clrMediumPurple,clrMe
   clrMediumSpringGreen,clrMediumTurquoise,clrMediumVioletRed,clrMidnightB
   clrNavajoWhite,clrNavy,clrOldLace,clrOlive,clrOliveDrab,clrOrange,clrOr
   clrPaleGreen,clrPaleTurquoise,clrPaleVioletRed,clrPapayaWhip,clrPeachPu
   clrPurple,clrRed,clrRosyBrown,clrRoyalBlue,clrSaddleBrown,clrSalmon,clr
   clrSienna,clrSilver,clrSkyBlue,clrSlateBlue,clrSlateGray,clrSnow,clrSpr
   clrThistle,clrTomato,clrTurquoise,clrViolet,clrWheat,clrWhite,clrWhiteS
  };
//+------------------------------------------------------------------+
//| Creating and initializing an edit object                         |
//+------------------------------------------------------------------+
void CreateColorBox(int x,int y,color c)
  {
//--- generate a name for a new edit object
   string name="ColorBox_"+(string)x+"_"+(string)y;
//--- create a new edit object
   if(!ObjectCreate(0,name,OBJ_EDIT,0,0,0))
     {
      Print("Cannot create: '",name,"'");
      return;
     }
//--- set coordinates, width and height
   ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x*X_SIZE);
   ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y*Y_SIZE);
   ObjectSetInteger(0,name,OBJPROP_XSIZE,X_SIZE);
   ObjectSetInteger(0,name,OBJPROP_YSIZE,Y_SIZE);
//--- set text color
   if(clrBlack==c) ObjectSetInteger(0,name,OBJPROP_COLOR,clrWhite);
   else            ObjectSetInteger(0,name,OBJPROP_COLOR,clrBlack);
//--- set background color
   ObjectSetInteger(0,name,OBJPROP_BGCOLOR,c);
//--- set text
   ObjectSetString(0,name,OBJPROP_TEXT,(string)c);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create 7x20 table of colored edit objects
   for(uint i=0;i<140;i++)
      CreateColorBox(i%7,i/7,ExtClr[i]);
  }
```

See also

Object Types, Object Properties

ObjectSetString

The function sets the value of the corresponding object property. The object property must be of the string type. There are 2 variants of the function.

Setting property value, without modifier:

```
bool  ObjectSetString(   long     chart_id,        // chart identifier
   string   object_name,     // object name
   int      prop_id,         // property
   string   prop_value       // value
   );
```

Setting a property value indicating the modifier:

```
bool  ObjectSetString(
   long     chart_id,        // chart identifier
   string   object_name,     // object name
   int      prop_id,         // property
   int      prop_modifier,   // modifier
   string   prop_value       // value
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

object_name

[in] Name of the object.

prop_id

[in] ID of the object property. The value can be one of the values of the ENUM_OBJECT_PROPERTY_STRING enumeration.

prop_modifier

[in] Modifier of the specified property. It denotes the number of the level in Fibonacci tools and in the graphical object Andrew's pitchfork. The numeration of levels starts from zero.

prop_value

[in] The value of the property.

Return Value

The function returns true only if the command to change properties of a graphical object has been sent to a chart successfully. Otherwise it returns false. To read more about the error call GetLastError().

TextSetFont

The function sets the font for displaying the text using drawing methods and returns the result of that operation. Arial font with the size -120 (12 pt) is used by default.

```
bool  TextSetFont(    const string  name,            // font name or path
   int            size,            // font size
   uint           flags=0,         // combination of flags
   int            orientation=0    // text slope angle
   );
```

Parameters

name

[in] Font name in the system or the name of the resource containing the font or the path to font file on the disk.

size

[in] The font size that can be set using positive and negative values. In case of positive values, the size of a displayed text does not depend on the operating system's font size settings. In case of negative values, the value is set in tenths of a point and the text size depends on the operating system settings ("standard scale" or "large scale"). See the Note below for more information about the differences between the modes.

flags=0

[in] Combination of flags describing font style.

orientation=0

[in] Text's horizontal inclination to X axis, the unit of measurement is 0.1 degrees. It means that *orientation=450* stands for inclination equal to 45 degrees.

Returned value

Returns true if the current font is successfully installed, otherwise false. Possible code errors:

- ERR_INVALID_PARAMETER(4003) - *name* presents NULL or "" (empty string),
- ERR_INTERNAL_ERROR(4001) - operating system error (for example, an attempt to create a non-existent font).

Note

If "::" is used in font name, the font is downloaded from EX4 resource. If *name* font name is specified with an extension, the font is downloaded from the file, if the path starts from "\" or "/", the file is searched relative to MQL4 directory. Otherwise, it is searched relative to the path of EX4 file which called TextSetFont() function.

The font size is set using positive or negative values. This fact defines the dependence of the text size from the operating system settings (size scale).

- If the size is specified by a positive number, this size is transformed into physical measurement units of a device (pixels) when changing a logical font into a physical one, and this size corresponds to the height of the symbol glyphs picked from the available fonts. This case is not recommended when the texts displayed by TextOut() function and the ones displayed by OBJ_LABEL ("Label") graphical object are to be used together on the chart.
- If the size is specified by a negative number, this number is supposed to be set in tenths of a logical point (-350 is equal to 35 logical points) and is divided by 10. An obtained value is then transformed into physical measurement units of a device (pixels) and corresponds to the absolute value of the height of a symbol picked from the available fonts. Multiply the font size specified in the object properties by -10 to make the size of a text on the screen similar to the one in OBJ_LABEL object.

The flags can be used as the combination of style flags with one of the flags specifying the font width. Flag names are shown below.

Flags for specifying font style

Flag	Description
FONT_ITALIC	Italic
FONT_UNDERLINE	Underline
FONT_STRIKEOUT	Strikeout

Flags for specifying font width

Flag
FW_DONTCARE
FW_THIN
FW_EXTRALIGHT
FW_ULTRALIGHT
FW_LIGHT

FW_NORMAL
FW_REGULAR
FW_MEDIUM
FW_SEMIBOLD
FW_DEMIBOLD
FW_BOLD
FW_EXTRABOLD
FW_ULTRABOLD
FW_HEAVY
FW_BLACK

See also

Resources, ResourceCreate(), ResourceSave(), TextOut()

TextOut

The function displays a text in a custom array (buffer) and returns the result of that operation. The array is designed to create the graphical resource.

```
bool  TextOut(   const string        text,          // displayed text
   int                 x,           // X coordinate
   int                 y,           // Y coordinate
   uint                anchor,      // anchor type
   uint                &data[],     // output buffer
   uint                width,       // buffer width in pixels
   uint                height,      // buffer height in pixels
   uint                color,       // text color
   ENUM_COLOR_FORMAT   color_format // color format for output
   );
```

Parameters

text

[in] Displayed text that will be written to the buffer. Only one-lined text is displayed.

x

[in] X coordinate of the anchor point of the displayed text.

y

[in] Y coordinate of the anchor point of the displayed text.

anchor

[in] The value out of the 9 pre-defined methods of the displayed text's anchor point location. The value is set by a combination of two flags flags of horizontal and vertical text align. Flag names are listed in the Note below.

data[]

[in] Buffer, in which text is displayed. The buffer is used to create the graphical resource.

width

[in] Buffer width in pixels.

height

[in] Buffer height in pixels.

color

[in] Text color.

`color_format`

[in] Color format is set by ENUM_COLOR_FORMAT enumeration value.

Returned value

Returns true if successful, otherwise false.

Note

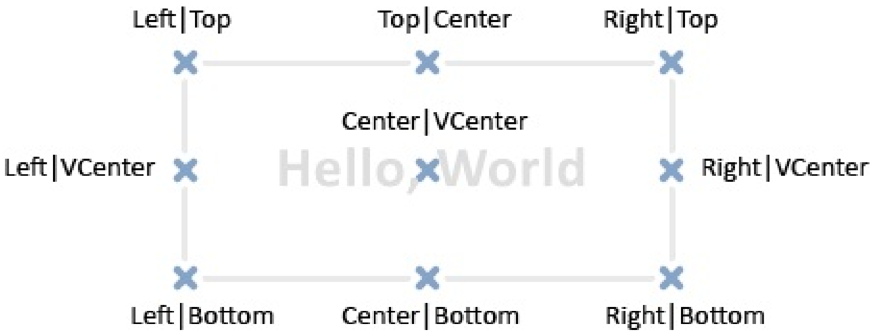
Anchor point specified by `anchor` is a combination of two flags of horizontal and vertical text align. Horizontal text align flags:

- TA_LEFT anchor point on the left side of the bounding box
- TA_CENTER horizontal anchor point is located at the center of the bounding box
- TA_RIGHT anchor point on the right side of the bounding box

Vertical text align flags:

- TA_TOP anchor point at the upper side of the bounding box
- TA_VCENTER vertical anchor point is located at the center of the bounding box
- TA_BOTTOM anchor point at the lower side of the bounding box

Possible combinations of flags and specified anchor points are shown in the image.



Example:

```
#property strict
//--- width and height of the canvas (used for drawing)
#define IMG_WIDTH  200
#define IMG_HEIGHT 200
//--- display the parameter window before launching the script
#property script_show_inputs
//--- enable to set color format
input ENUM_COLOR_FORMAT clr_format=COLOR_FORMAT_XRGB_NOALPHA;
//--- drawing array (buffer)
uint ExtImg[IMG_WIDTH*IMG_HEIGHT];
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- create OBJ_BITMAP_LABEL object for drawing
   ObjectCreate(0,"CLOCK",OBJ_BITMAP_LABEL,0,0,0);
//--- specify the name of the graphical resource for writing in CLOCK obje
   ObjectSetString(0,"CLOCK",OBJPROP_BMPFILE,"::IMG");

//--- auxiliary variables
   double a;             // arrow corner
   uint   nm=2700;       // minute corner
   uint   nh=2700*12;    // hour counter
   uint   w,h;           // variables for receiving text string sizes
   int    x,y;           // variables for calculation of the current coordi

//--- rotate clock hands in an infinite loop, till the script is stopped
   while(!IsStopped())
     {
      //--- clear the clock drawing buffer array
      ArrayFill(ExtImg,0,IMG_WIDTH*IMG_HEIGHT,0);
      //--- set the font for drawing digits for the clock-face
      TextSetFont("Arial",-200,FW_EXTRABOLD,0);
      //--- draw the clock-face
      for(int i=1;i<=12;i++)
        {
         //--- receive the size of the current hour on the clock-face
         TextGetSize(string(i),w,h);
         //--- calculate the coordinates of the current hour on the clock-
         a=-((i*300)%3600*M_PI)/1800.0;
         x=IMG_WIDTH/2-int(sin(a)*80+0.5+w/2);
         y=IMG_HEIGHT/2-int(cos(a)*80+0.5+h/2);
         //--- output the hour on the clock-face to ExtImg[] buffer
         TextOut(string(i),x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,
        }
      //--- now, specify the font for drawing the minute hand
      TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nm%3600));
      //--- receive the size of the minute hand
      TextGetSize("----->",w,h);
      //--- calculate the coordinates of the minute hand on the clock-face
      a=-(nm%3600*M_PI)/1800.0;
      x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
      y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
      //--- output of the minute hand to the clock-face in ExtImg[]buffer
      TextOut("----->",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFF

      //--- now, set the font for drawing the minute hand
      TextSetFont("Arial",-200,FW_EXTRABOLD,-int(nh/12%3600));
      TextGetSize("==>",w,h);
      //--- calculate the coordinates of the hour hand on the clock-face
      a=-(nh/12%3600*M_PI)/1800.0;
      x=IMG_WIDTH/2-int(sin(a)*h/2+0.5);
      y=IMG_HEIGHT/2-int(cos(a)*h/2+0.5);
      //--- output of the hour hand on the clock-face to ExtImg[] buffer
      TextOut("==>",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFF

      //--- update the graphical resource
      ResourceCreate("::IMG",ExtImg,IMG_WIDTH,IMG_HEIGHT,0,0,IMG_WIDTH,clr
      //--- forced chart update
      ChartRedraw();

      //--- increase hour and minute counters
      nm+=60;
      nh+=60;
      //--- keeping a short pause between the frames
      Sleep(10);
     }
//--- delete CLOCK object when completing the script's operation
   ObjectDelete(0,"CLOCK");
//---
  }
```

See also

Resources, ResourceCreate(), ResourceSave(), TextGetSize(), TextSetFont()

TextGetSize

The function returns the line width and height at the current font settings.

```
bool  TextGetSize(    const string    text,     // text string
   uint&           width,    // buffer width in pixels
   uint&           height    // buffer height in pixels
   );
```

Parameters

text

[in] String, for which length and width should be obtained.

width

[out] Input parameter for receiving width.

height

[out] Input parameter for receiving height.

Returned value

Returns true if successful, otherwise false. Possible code errors:

· ERR_INTERNAL_ERROR(4001) - operating system error.

See also

Resources, ResourceCreate(), ResourceSave(), TextSetFont(), TextOut()

ObjectDescription

Returns the object description.

```
string  ObjectDescription(    string    object_name    // object name
   );
```

Parameters

object_name

[in] Object name.

Returned value

Object description. For objects of OBJ_TEXT and OBJ_LABEL types, the text drawn by these objects will be returned. To get the detailed error information, one has to call the GetLastError() function.

Example:

```
  // writing the chart objects 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= "+ObjectDescrip
       }
     FileClose(handle);
    }
```

See also

ObjectSetText()

ObjectGet

Returns the value of the specified object property.

```
double  ObjectGet(    string    object_name,    // object name
   int        index           // object property
   );
```

Parameters

object_name

[in] Object name.

index

[in] Object property index. It can be any of the Object properties enumeration values.

Returned value

The value of the specified object property. To check errors, one has to call the GetLastError() function.

Example:

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

See also

ObjectSet()

ObjectGetFiboDescription

Returns the level description of a Fibonacci object.

```
string  ObjectGetFiboDescription(    string    object_name,    // object nam
   int       index            // level index
   );
```

Parameters

object_name

[in] Fibonacci object name.

index

[in] Index of the Fibonacci level (0-31).

Returned value

The level description of a Fibonacci object. To get the detailed error information, one has to call the GetLastError() function.

Note

The amount of Fibonacci levels depends on the object type. The maximum amount of Fibonacci levels is 32.

Example:

```
#include <stdlib.mqh>
  ...
  string text;
  for(int i=0;i<32;i++)
    {
     text=ObjectGetFiboDescription(MyObjectName,i);
     //---- check if your object has less than 32 levels
     if(GetLastError()!=ERR_NO_ERROR) break;
     Print(MyObjectName," level index: ",i," description: ",text);
    }
```

See also

ObjectSetFiboDescription()

ObjectGetShiftByValue

The function calculates and returns bar index (shift related to the current bar) for the given price.

```
int  ObjectGetShiftByValue(    string    object_name,    // object name
   double    value            // price
   );
```

Parameters

object_name

[in] Object name.

value

[in] Price value.

Returned value

The function calculates and returns bar index (shift related to the current bar) for the given price. The bar index is calculated by the first and second coordinates using a linear equation. Applied to trendlines and similar objects. To get the detailed error information, one has to call the GetLastError() function.

Example:

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

See also

ObjectGetValueByShift()

ObjectGetValueByShift

The function calculates and returns the price value for the specified bar (shift related to the current bar).

```
double  ObjectGetValueByShift(    string    object_name,    // object name
   int        shift           // bar index
   );
```

Parameters

object_name

[in] Object name.

shift

[in] Bar index.

Returned value

The function calculates and returns the price value for the specified bar (shift related to the current bar). The price value is calculated by the first and second coordinates using a linear equation. Applied to trendlines and similar objects. To get the detailed error information, one has to call the GetLastError() function.

Example:

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

See also

ObjectGetShiftByValue()

ObjectSet

Changes the value of the specified object property.

```
bool  ObjectSet(   string   object_name,    // object name
   int       index,          // property index
   double    value           // value
   );
```

Parameters

object_name

[in] Object name.

index

[in] Object property index. It can be any of object properties enumeration values.

value

[in] New value of the given property.

Returned value

If the function succeeds, the returned value will be true, otherwise it returns false. To get the detailed error information, one has to call the GetLastError() function.

Example:

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

See also

ObjectGet()

ObjectSetFiboDescription

The function sets a new description to a level of a Fibonacci object.

```
bool  ObjectSetFiboDescription(    string   object_name,    // object name
   int      index,         // level index
   string   text           // new description
   );
```

Parameters

object_name

[in] Object name.

index

[in] Index of the Fibonacci level (0-31).

text

[in] New description of the level.

Returned value

The function returns true if successful, otherwise false. To get the detailed error information, one has to call the GetLastError() function.

Note

The amount of Fibonacci levels depends on the object type. The maximum amount of Fibonacci levels is 32.

Example:

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

See also

ObjectGetFiboDescription()

ObjectSetText

The function changes the object description.

```
bool  ObjectSetText(   string   object_name,          // object name
   string   text,                  // description
   int      font_size=0,           // font size
   string   font_name=NULL,        // font name
   color    text_color=clrNONE     // text color
   );
```

Parameters

object_name

[in] Object name.

text

[in] A text describing the object.

font_size=0

[in] Font size in points.

font_name=NULL

[in] Font name.

text_color=clrNONE

[in] Font color.

Returned value

Changes the object description. If the function succeeds, the returned value will be true, otherwise false. To get the detailed error information, one has to call the GetLastError() function.

Note

For objects of OBJ_TEXT and OBJ_LABEL, this description is shown as a text line in the chart. Parameters of font_size, font_name and text_color are used for objects of OBJ_TEXT and OBJ_LABEL only. For objects of other types, these parameters are ignored.

Example:

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

See also

ObjectDescription()

ObjectType

The function returns the object type value.

```
int  ObjectType(    string    object_name    // object name
   );
```

Parameters

object_name

[in] Object name.

Returned value

The function returns the object type value. To get the detailed error information, one has to call the GetLastError() function.

Example:

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

See also

Object types

Technical Indicator Functions

A group of functions intended for calculation of standard and custom indicators.

For an Expert Advisor (or any other MQL4 program) to take up the value of any indicator, it is not necessary that this indicator is present in the chart. The requested indicator will be loaded and calculated in the thread of the module that has called it.

Any indicator can be calculated on the data of not only current chart, but also on the data of any available symbol/period. If data (symbol name and/or timeframe differ from the current ones) are requested from another chart, the situation is possible that the corresponding chart was not opened in the client terminal and the necessary data must be requested from the server. In this case, error ERR_HISTORY_WILL_UPDATED (4066 - the requested history data are under updating) will be placed in the last_error variable, and one will has to re-request (see example of ArrayCopySeries()).

All indicator functions have at least 2 parameters - symbol and period. The NULL value of the symbol means the current symbol, the 0 value of the period means the current timeframe.

Function	Returns the indicator value
iAC	Accelerator Oscillator
iAD	Accumulation/Distribution
iADX	Average Directional Index
iAlligator	Alligator
iAO	Awesome Oscillator
iATR	Average True Range
iBearsPower	Bears Power
iBands	Bollinger Bands®
iBandsOnArray	Calculation of Bollinger Bands® indicator on data, stored in a numeric array
iBullsPower	Bulls Power
iCCI	Commodity Channel Index
iCCIOnArray	Calculation of Commodity Channel Index indicator on data, stored in a numeric array

iCustom	Custom indicator
iDeMarker	DeMarker
iEnvelopes	Envelopes
iEnvelopesOnArray	Calculation of Envelopes indicator on data, stored in a numeric array
iForce	Force Index
iFractals	Fractals
iGator	Gator Oscillator
iIchimoku	Ichimoku Kinko Hyo
iBWMFI	Market Facilitation Index by Bill Williams
iMomentum	Momentum
iMomentumOnArray	Calculation of Momentum indicator on data, stored in a numeric array
iMFI	Money Flow Index
iMA	Moving Average
iMAOnArray	Calculation of Moving Average indicator on data, stored in a numeric array
iOsMA	Moving Average of Oscillator (MACD histogram)
iMACD	Moving Averages Convergence-Divergence
iOBV	On Balance Volume
iSAR	Parabolic Stop And Reverse System
iRSI	Relative Strength Index
iRSIOnArray	Calculation of Momentum indicator on data, stored in a numeric array
iRVI	Relative Vigor Index
iStdDev	Standard Deviation
iStdDevOnArray	Calculation of Standard Deviation indicator on data, stored in a numeric array
iStochastic	Stochastic Oscillator
iWPR	Williams' Percent Range

iAC

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

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

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Bill Williams' Accelerator/Decelerator oscillator.

Example:

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

iAD

Calculates the Accumulation/Distribution indicator and returns its value.

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

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Accumulation/Distribution indicator.

Example:

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

iADX

Calculates the Average Directional Movement Index indicator and returns its value.

```
double  iADX(    string        symbol,         // symbol
   int           timeframe,     // timeframe
   int           period,        // averaging period
   int           applied_price, // applied price
   int           mode,          // line index
   int           shift          // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period for calculation.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

mode

[in] Indicator line index. It can be any of the Indicators line identifiers enumeration value. (0 - MODE_MAIN, 1 - MODE_PLUSDI, 2 - MODE_MINUSDI).

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Average Directional Movement Index indicator.

Example:

```
  if(iADX(NULL,0,14,PRICE_HIGH,MODE_MAIN,0)>iADX(NULL,0,14,PRICE_HIGH,MODE
```

iAlligator

Calculates the Alligator indicator and returns its value.

```
double  iAlligator(    string       symbol,            // symbol
   int          timeframe,         // timeframe
   int          jaw_period,        // Jaw line averaging period
   int          jaw_shift,         // Jaw line shift
   int          teeth_period,      // Teeth line averaging period
   int          teeth_shift,       // Teeth line shift
   int          lips_period,       // Lips line averaging period
   int          lips_shift,        // Lips line shift
   int          ma_method,         // averaging method
   int          applied_price,     // applied price
   int          mode,              // line index
   int          shift              // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

jaw_period

[in] Blue line averaging period (Alligator's Jaw).

jaw_shift

[in] Blue line shift relative to the chart.

teeth_period

[in] Red line averaging period (Alligator's Teeth).

teeth_shift

[in] Red line shift relative to the chart.

lips_period

[in] Green line averaging period (Alligator's Lips).

lips_shift

[in] Green line shift relative to the chart.

ma_method

[in] MA method. It can be any of Moving Average methods. It can be any of ENUM_MA_METHOD enumeration values.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

mode

[in] Data source, identifier of the indicator line. 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

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Alligator indicator.

Example:

```
  double jaw_val=iAlligator(NULL,0,13,8,8,5,5,3,MODE_SMMA,PRICE_MEDIAN,MOD
```

iAO

Calculates the Awesome oscillator and returns its value.

```
double  iAO(    string       symbol,     // symbol
   int          timeframe,   // timeframe
   int          shift        // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Awesome oscillator indicator.

Example:

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

iATR

Calculates the Average True Range indicator and returns its value.

```
double  iATR(    string       symbol,     // symbol
   int          timeframe,   // timeframe
   int          period,      // averaging period
   int          shift        // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Average True Range indicator.

Example:

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

iBearsPower

Calculates the Bears Power indicator and returns its value.

```
double  iBearsPower(    string       symbol,           // symbol
   int          timeframe,        // timeframe
   int          period,           // averaging period
   int          applied_price,    // applied price
   int          shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Bears Power indicator.

Example:

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

iBands

Calculates the Bollinger Bands® indicator and returns its value.

```
double  iBands(    string        symbol,            // symbol
   int            timeframe,        // timeframe
   int            period,           // averaging period
   double         deviation,        // standard deviations
   int            bands_shift,      // bands shift
   int            applied_price,    // applied price
   int            mode,             // line index
   int            shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period to calculate the main line.

deviation

[in] Number of standard deviations from the main line.

bands_shift

[in] The indicator shift relative to the chart.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

mode

[in] Indicator line index. It can be any of the Indicators line identifiers enumeration value (0 - MODE_MAIN, 1 - MODE_UPPER, 2 - MODE_LOWER).

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Bollinger Bands® indicator.

Example:

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

iBandsOnArray

Calculates the Bollinger Bands® indicator on data, stored in array, and returns its value.

```
double  iBandsOnArray(    double        array[],          // array with dat
   int           total,             // number of elements
   int           period,            // averaging period
   double        deviation,         // deviation
   int           bands_shift,       // bands shift
   int           mode,              // line index
   int           shift              // shift
   );
```

Parameters

array[]

[in] Array with data.

total

[in] The number of items to be counted. 0 means the whole array.

period

[in] Averaging period to calculate the main line.

deviation

[in] Number of standard deviations from the main line.

bands_shift

[in] The indicator shift relative to the chart.

mode

[in] Indicator line index. It can be any of the Indicators line identifiers enumeration value (0 - MODE_MAIN, 1 - MODE_UPPER, 2 - MODE_LOWER).

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Bollinger Bands® indicator, calculated on data, stored in array[].

Note

Unlike iBands(...), the iBandsOnArray() function does not take data by symbol name, timeframe, the applied price. The price data must be previously prepared. The indicator is calculated from left to right. To access to the array elements as to a series array (i.e., from right to left), one has to use the ArraySetAsSeries() function.

Example:

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

iBullsPower

Calculates the Bulls Power indicator and returns its value.

```
double  iBullsPower(    string        symbol,           // symbol
   int           timeframe,         // timeframe
   int           period,            // averaging period
   int           applied_price,     // applied price
   int           shift              // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period for calculation.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Bulls Power indicator.

Example:

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

iCCI

Calculates the Commodity Channel Index indicator and returns its value.

```
double  iCCI(    string        symbol,           // symbol
   int          timeframe,        // timeframe
   int          period,           // averaging period
   int          applied_price,    // applied price
   int          shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period for calculation.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Commodity Channel Index indicator.

Example:

```
  if(iCCI(Symbol(),0,12,PRICE_TYPICAL,0)>iCCI(Symbol(),0,20,PRICE_TYPICAL,
```

iCCIOnArray

Calculates the Commodity Channel Index indicator on data, stored in array, and returns its value.

```
double  iCCIOnArray(    double        array[],          // array with data
   int          total,            // number of elements
   int          period,           // averaging period
   int          shift             // shift
   );
```

Parameters

array[]

[in] Array with data.

total

[in] The number of items to be counted. 0 means the whole array.

period

[in] Averaging period for calculation.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Commodity Channel Index indicator, calculated on data, stored in array[].

Note

Unlike iCCI(...), the iCCIOnArray() function does not take data by symbol name, timeframe, the applied price. The price data must be previously prepared. The indicator is calculated from left to right. To access to the array elements as to a series array (i.e., from right to left), one has to use the ArraySetAsSeries() function.

Example:

```
   if(iCCIOnArray(ExtBuffer,total,12,0)>iCCIOnArray(ExtBuffer,total,20,0))
```

iCustom

Calculates the specified custom indicator and returns its value.

```
double  iCustom(    string       symbol,           // symbol
   int          timeframe,        // timeframe
   string       name,             // path/name of the custom indicator com
   ...                            // custom indicator input parameters (if
   int          mode,             // line index
   int          shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

name

[in] Custom indicator compiled program name, relative to the root indicators directory (MQL4/Indicators/). If the indicator is located in subdirectory, for example, in MQL4/Indicators/**Examples**, its name must be specified as "**Examples**\\indicator_name" (double backslash "\\"must be specified as separator instead of a single one).

...

[in] Custom indicator input-parameters, separated by commas.

The passed parameters and their order must correspond with the declaration order and the type of extern variables of the custom indicator. If the values of input parameters is not specified, the default values will be used.

mode

[in] Line index. Can be from 0 to 7 and must correspond with the index, specified in call of the SetIndexBuffer() function.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the specified custom indicator. The custom indicator must be compiled (*.EX4 file) and be in the terminal_directory\MQL4\Indicators\ directory.

Example:

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

iDeMarker

Calculates the DeMarker indicator and returns its value.

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

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period for calculation.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the DeMarker indicator.

Example:

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

iEnvelopes

Calculates the Envelopes indicator and returns its value.

```
double  iEnvelopes(    string        symbol,            // symbol
   int          timeframe,        // timeframe
   int          ma_period,        // MA averaging period
   int          ma_method,        // MA averaging method
   int          ma_shift,         // moving average shift
   int          applied_price,    // applied price
   double       deviation,        // deviation (in percents)
   int          mode,             // line index
   int          shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

ma_period

[in] Averaging period for calculation of the main line.

ma_method

[in] Moving Average method. It can be any of ENUM_MA_METHOD enumeration values.

ma_shift

[in] MA shift. Indicator line offset relate to the chart by timeframe.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

deviation

[in] Percent deviation from the main line.

mode

[in] Indicator line index. It can be any of Indicators line identifiers enumeration value (0 - MODE_MAIN, 1 - MODE_UPPER, 2 - MODE_LOWER).

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Envelopes indicator.

Example:

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

iEnvelopesOnArray

Calculates the Envelopes indicator on data, stored in array, and returns its value.

```
double  iEnvelopesOnArray(    double       array[],          // array with
   int          total,            // number of elemements
   int          ma_period,        // MA averaging period
   int          ma_method,        // MA averaging method
   int          ma_shift,         // MA shift
   double       deviation,        // deviation (in percents)
   int          mode,             // line index
   int          shift             // shift
   );
```

Parameters

array[]

[in] Array with data.

total

[in] The number of items to be counted. 0 means the whole array.

ma_period

[in] Averaging period for calculation of the main line.

ma_method

[in] Moving Average method. It can be any of ENUM_MA_METHOD enumeration values.

ma_shift

[in] MA shift. Indicator line offset relate to the chart by timeframe.

deviation

[in] Percent deviation from the main line.

mode

[in] Indicator line index. It can be any of Indicators line identifiers enumeration value (0 - MODE_MAIN, 1 - MODE_UPPER, 2 - MODE_LOWER).

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Envelopes indicator, calculated on data, stored in array[].

Note

Unlike iEnvelopes(...), the iEnvelopesOnArray() function does not take data by symbol name, timeframe, the applied price. The price data must be previously prepared. The indicator is calculated from left to right. To access to the array elements as to a series array (i.e., from right to left), one has to use the ArraySetAsSeries() function.

Example:

```
  double val=iEnvelopesOnArray(ExtBuffer,10,13,MODE_SMA,0,0.2,MODE_UPPER,0
```

iForce

Calculates the Force Index indicator and returns its value.

```
double  iForce(    string       symbol,           // symbol
   int          timeframe,        // timeframe
   int          period,           // averaging period
   int          ma_method,        // averaging method
   int          applied_price,    // applied price
   int          shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period for calculation.

ma_method

[in] Moving Average method. It can be any of ENUM_MA_METHOD enumeration values.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Force Index indicator.

Example:

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

iFractals

Calculates the Fractals indicator and returns its value.

```
double  iFractals(    string            symbol,           // symbol
   int             timeframe,        // timeframe
   int             mode,             // line index
   int             shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

mode

[in] Indicator line index. It can be any of the Indicators line identifiers enumeration value.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Fractals indicator.

Example:

```
  double val=iFractals(NULL,0,MODE_UPPER,3);
```

iGator

Calculates the Gator oscillator and returns its value.

```
double  iGator(    string       symbol,            // symbol
   int          timeframe,          // timeframe
   int          jaw_period,         // Jaw line period
   int          jaw_shift,          // Jaw line shift
   int          teeth_period,       // Teeth line period
   int          teeth_shift,        // Teeth line shift
   int          lips_period,        // Lips line period
   int          lips_shift,         // Lips line shift
   int          ma_method,          // MA averaging method
   int          applied_price,      // applied price
   int          mode,               // line index
   int          shift               // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

jaw_period

[in] Blue line averaging period (Alligator's Jaw).

jaw_shift

[in] Blue line shift relative to the chart.

teeth_period

[in] Red line averaging period (Alligator's Teeth).

teeth_shift

[in] Red line shift relative to the chart.

lips_period

[in] Green line averaging period (Alligator's Lips).

lips_shift

[in] Green line shift relative to the chart.

ma_method

[in] MA method. It can be any of Moving Average method enumeration value.

applied_price

[in] Applied price. It can be any of Applied price enumeration values.

mode

[in] Indicator line index. It can be any of Indicators line identifiers enumeration value.

MODE_GATORJAW - blue line (Jaw line),
MODE_GATORTEETH - red line (Teeth line),
MODE_GATORLIPS - green line (Lips line).

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Gator oscillator.

Note

The oscillator displays the difference between the Alligator red and blue lines (the upper histogram) and that between red and green lines (the lower histogram).

Example:

```
  double diff=iGator(NULL,0,13,8,8,5,5,3,MODE_SMMA,PRICE_MEDIAN,MODE_UPPER
```

iIchimoku

Calculates the Ichimoku Kinko Hyo indicator and returns its value.

```
double  iIchimoku(   string        symbol,            // symbol
   int          timeframe,         // timeframe
   int          tenkan_sen,        // period of Tenkan-sen line
   int          kijun_sen,         // period of Kijun-sen line
   int          senkou_span_b,     // period of Senkou Span B line
   int          mode,              // line index
   int          shift              // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

tenkan_sen

[in] Tenkan Sen averaging period.

kijun_sen

[in] Kijun Sen averaging period.

senkou_span_b

[in] Senkou SpanB averaging period.

mode

[in] Source of data. It can be one of the Ichimoku Kinko Hyo mode enumeration (1 - MODE_TENKANSEN, 2 - MODE_KIJUNSEN, 3 - MODE_SENKOUSPANA, 4 - MODE_SENKOUSPANB, 5 - MODE_CHIKOUSPAN).

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Ichimoku Kinko Hyo indicator.

Example:

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

iBWMFI

Calculates the Market Facilitation Index indicator and returns its value.

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

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Market Facilitation Index indicator.

Example:

```
  double val=iBWMFI(NULL,10,0);
```

iMomentum

Calculates the Momentum indicator and returns its value.

```
double  iMomentum(    string         symbol,           // symbol
   int           timeframe,        // timeframe
   int           period,           // averaging period
   int           applied_price,    // applied price
   int           shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period (amount of bars) for calculation of price changes.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Momentum indicator.

Example:

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

iMomentumOnArray

Calculates the Momentum indicator on data, stored in array, and returns its value.

```
double  iMomentumOnArray(    double        array[],          // array with
   int           total,             // number of elements
   int           period,            // averaging period
   int           shift              // shift
   );
```

Parameters

array[]

[in] Array with data.

total

[in] The number of items to be counted. 0 means the whole array.

period

[in] Averaging period (amount of bars) for calculation of price changes.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Momentum indicator, calculated on data, stored in array[].

Note

Unlike iMomentum(...), the iMomentumOnArray() function does not take data by symbol name, timeframe, the applied price. The price data must be previously prepared. The indicator is calculated from left to right. To access to the array elements as to a series array (i.e., from right to left), one has to use the ArraySetAsSeries() function.

Example:

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

iMFI

Calculates the Money Flow Index indicator and returns its value.

```
double  iMFI(    string        symbol,     // symbol
   int           timeframe,   // timeframe
   int           period,      // averaging period
   int           shift        // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Period (amount of bars) for calculation of the indicator.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Money Flow Index indicator.

Example:

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

iMA

Calculates the Moving Average indicator and returns its value.

```
double  iMA(    string       symbol,           // symbol
   int          timeframe,        // timeframe
   int          ma_period,        // MA averaging period
   int          ma_shift,         // MA shift
   int          ma_method,        // averaging method
   int          applied_price,    // applied price
   int          shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

ma_period

[in] Averaging period for calculation.

ma_shift

[in] MA shift. Indicators line offset relate to the chart by timeframe.

ma_method

[in] Moving Average method. It can be any of ENUM_MA_METHOD enumeration values.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Moving Average indicator.

Example:

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

iMAOnArray

Calculates the Moving Average indicator on data, stored in array, and returns its value.

```
double  iMAOnArray(    double        array[],           // array with data
   int          total,            // number of elements
   int          ma_period,        // MA averaging period
   int          ma_shift,         // MA shift
   int          ma_method,        // MA averaging method
   int          shift             // shift
   );
```

Parameters

array[]

[in] Array with data.

total

[in] The number of items to be counted. 0 means the whole array.

ma_period

[in] Averaging period for calculation.

ma_shift

[in] MA shift. Indicators line offset relate to the chart by timeframe.

ma_method

[in] Moving Average method. It can be any of ENUM_MA_METHOD enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Moving Average indicator, calculated on data, stored in array[].

Note

Unlike iMA(...), the iMAOnArray() function does not take data by symbol name, timeframe, the applied price. The price data must be previously prepared. The indicator is calculated from left to right. To access to the array elements as to a series array (i.e., from right to left), one has to use the ArraySetAsSeries() function.

Example:

```
   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");
```

iOsMA

Calculates the Moving Average of Oscillator indicator and returns its value.

```
double  iOsMA(    string        symbol,            // symbol
   int           timeframe,         // timeframe
   int           fast_ema_period,   // Fast EMA period
   int           slow_ema_period,   // Slow EMA period
   int           signal_period,     // Signal line period
   int           applied_price,     // applied price
   int           shift              // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

fast_ema_period

[in] Fast EMA averaging period.

slow_ema_period

[in] Slow EMA averaging period.

signal_period

[in] Signal line averaging period.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Moving Average of Oscillator.

Example:

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

iMACD

Calculates the Moving Averages Convergence/Divergence indicator and returns its value.

```
double  iMACD(    string        symbol,            // symbol
   int          timeframe,          // timeframe
   int          fast_ema_period,    // Fast EMA period
   int          slow_ema_period,    // Slow EMA period
   int          signal_period,      // Signal line period
   int          applied_price,      // applied price
   int          mode,               // line index
   int          shift               // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

fast_ema_period

[in] Fast EMA averaging period.

slow_ema_period

[in] Slow EMA averaging period.

signal_period

[in] Signal line averaging period.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

mode

[in] Indicator line index. It can be one of the Indicators line identifiers enumeration values (0-MODE_MAIN, 1-MODE_SIGNAL).

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Moving Average of Oscillator indicator.

Note

In some systems it is called MACD Histogram and plotted as two lines. In MetaTrader 4 client terminal MACD is plotted as histogram.

Example:

```
  if(iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,0)>iMACD(NULL,0,12,26,9,PR
```

iOBV

Calculates the On Balance Volume indicator and returns its value.

```
double  iOBV(    string        symbol,            // symbol
   int          timeframe,         // timeframe
   int          applied_price,     // applied price
   int          shift              // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the On Balance Volume indicator.

Example:

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

iSAR

Calculates the Parabolic Stop and Reverse system indicator and returns its value.

```
double  iSAR(    string       symbol,            // symbol
   int          timeframe,         // timeframe
   double       step,              // price increment step - acceleration
   double       maximum,           // maximum value of step
   int          shift              // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

step

[in] The step of price increment, usually 0.02.

maximum

[in] The maximum step, usually 0.2.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Parabolic Stop and Reverse system indicator.

Example:

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

iRSI

Calculates the Relative Strength Index indicator and returns its value.

```
double  iRSI(    string       symbol,            // symbol
   int          timeframe,         // timeframe
   int          period,            // period
   int          applied_price,     // applied price
   int          shift              // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period for calculation.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Relative Strength Index indicator.

Example:

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

iRSIOnArray

Calculates the Relative Strength Index indicator on data, stored in array, and returns its value.

```
double  iRSIOnArray(    double        array[],          // array with data
   int          total,            // number of elements
   int          period,           // averaging period
   int          shift             // shift
   );
```

Parameters

array[]

[in] Array with data.

total

[in] The number of items to be counted. 0 means the whole array.

period

[in] Averaging period for calculation.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Relative Strength Index indicator, calculated on data, stored in array[].

Note

Unlike iRSI(...), the iRSIOnArray() does not take data by symbol name, timeframe, the applied price. The price data must be previously prepared. The indicator is calculated from left to right. To access to the array elements as to a series array (i.e., from right to left), one has to use the ArraySetAsSeries() function.

Example:

```
   if(iRSIOnArray(ExtBuffer,1000,14,0)>iRSI(NULL,0,14,PRICE_CLOSE,1)) retur
```

iRVI

Calculates the Relative Vigor Index indicator and returns its value.

```
double  iRVI(    string         symbol,            // symbol
   int          timeframe,        // timeframe
   int          period,           // averaging period
   int          mode,             // line index
   int          shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period for calculation.

mode

[in] Indicator line index. It can be any of Indicators line identifiers enumeration value.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Relative Vigor Index indicator.

Example:

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

iStdDev

Calculates the Standard Deviation indicator and returns its value.

```
double  iStdDev(    string       symbol,           // symbol
   int          timeframe,        // timeframe
   int          ma_period,        // MA averaging period
   int          ma_shift,         // MA shift
   int          ma_method,        // MA averaging method
   int          applied_price,    // applied price
   int          shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

ma_period

[in] Moving Average period.

ma_shift

[in] Moving Average shift.

ma_method

[in] Moving Average method. It can be any of ENUM_MA_METHOD enumeration values.

applied_price

[in] Applied price. It can be any of ENUM_APPLIED_PRICE enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Standard Deviation indicator.

Example:

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

iStdDevOnArray

Calculates the Standard Deviation indicator on data, stored in array, and returns its value.

```
double  iStdDevOnArray(   double       array[],          // array with da
   int            total,           // number of elements
   int            ma_period,       // MA averaging period
   int            ma_shift,        // MA shift
   int            ma_method,       // MA averaging method
   int            shift            // shift
   );
```

Parameters

array[]

[in] Array with data.

total

[in] The number of items to be counted. 0 means the whole array.

ma_period

[in] Moving Average period.

ma_shift

[in] Moving Average shift.

ma_method

[in] Moving Average method. It can be any of ENUM_MA_METHOD enumeration values.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Standard Deviation indicator, calculated on data, stored in array[].

Note

Unlike iStdDev(...), the iStdDevOnArray() does not take data by symbol name, timeframe, the applied price. The price data must be previously prepared. The indicator is calculated from left to right. To access to the array elements as to a series array (i.e., from right to left), one has to use the ArraySetAsSeries() function.

Example:

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

iStochastic

Calculates the Stochastic Oscillator and returns its value.

```
double  iStochastic(   string         symbol,           // symbol
   int          timeframe,        // timeframe
   int          Kperiod,          // K line period
   int          Dperiod,          // D line period
   int          slowing,          // slowing
   int          method,           // averaging method
   int          price_field,      // price (Low/High or Close/Close)
   int          mode,             // line index
   int          shift             // shift
   );
```

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

Kperiod

[in] Period of the %K line.

Dperiod

[in] Period of the %D line.

slowing

[in] Slowing value.

method

[in] Moving Average method. It can be any of ENUM_MA_METHOD enumeration values.

price_field

[in] Price field parameter. Can be one of this values: 0 - Low/High or 1 - Close/Close.

mode

[in] Indicator line index. It can be any of the Indicators line identifiers enumeration value (0 - MODE_MAIN, 1 - MODE_SIGNAL).

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Stochastic Oscillator.

Example:

```
  if(iStochastic(NULL,0,5,3,3,MODE_SMA,0,MODE_MAIN,0)>iStochastic(NULL,0,5
```

iWPR

Calculates the Larry Williams' Percent Range and returns its value.

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

Parameters

symbol

[in] Symbol name on the data of which the indicator will be calculated. NULL means the current symbol.

timeframe

[in] Timeframe. It can be any of ENUM_TIMEFRAMES enumeration values. 0 means the current chart timeframe.

period

[in] Averaging period for calculation.

shift

[in] Index of the value taken from the indicator buffer (shift relative to the current bar the given amount of periods ago).

Returned value

Numerical value of the Larry Williams' Percent Range indicator.

Example:

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

Event Functions

This group contains functions for working with custom events and timer events. Besides this group, there are special functions for handling predefined events.

Function	Action
EventSetMillisecondTimer	Launches event generator of the high-resolution timer with a period less than 1 second for the current chart
EventSetTimer	Starts the timer event generator with the specified periodicity for the current chart
EventKillTimer	Stops the generation of events by the timer in the current chart
EventChartCustom	Generates a custom event for the specified chart

See also

Types of Chart Events

EventSetMillisecondTimer

The function indicates to the client terminal that timer events should be generated at intervals less than one second for this Expert Advisor or indicator.

```
bool  EventSetMillisecondTimer(    int  milliseconds      // number of mil
   );
```

Parameters

milliseconds

[in] Number of milliseconds defining the frequency of timer events.

Returned value

In case of successful execution, returns true, otherwise - false. To receive an error code, GetLastError() function should be called.

Note

This feature is designed for the cases when high-resolution timer is required. In other words, timer events should be received more frequently than once per second. If a conventional timer with the period of several seconds is enough for you, use EventSetTimer().

Usually, this function should be called from OnInit() function or in class constructor. To handle events coming from the timer, an Expert Advisor or an indicator should have OnTimer() function.

Each Expert Advisor and each indicator work with its own timer receiving events solely from this timer. During mql4 application shutdown, the timer is forcibly destroyed in case it has been created but has not been disabled by EventKillTimer() function.

Only one timer can be launched for each program. Each mql4 application and chart have their own queue of events where all newly arrived events are placed. If the queue already contains Timer event or this event is in the processing stage, then the new Timer event is not added to mql4 application queue.

EventSetTimer

The function indicates to the client terminal, that for this indicator or Expert Advisor, events from the timer must be generated with the specified periodicity.

```
bool  EventSetTimer(    int   seconds        // number of seconds
   );
```

Parameters

seconds

[in] Number of seconds that determine the frequency of the timer event occurrence.

Return Value

In case of success returns true, otherwise false. In order to get an error code, the GetLastError() function should be called.

Note

Normally, this function must be called from the OnInit() function or from a class constructor. In order to handle events coming from the timer, the Expert Advisor must have the OnTimer() function.

Every Expert Advisor, as well as every indicator works with its own timer and receives events only from it. As soon as a mql4 program stops operating, the timer is destroyed forcibly if it was created but hasn't been disabled by the EventKillTimer() function.

For each program no more than one timer can be run. Each mql4 program and each chart has its own queue of events, in which all the newly received events are placed. If the Timer event is present in the queue or is being processed, the new Timer event will not be placed in the queue of the mql4 program.

EventKillTimer

Specifies the client terminal that is necessary to stop the generation of events from Timer.

```
void  EventKillTimer();
```

Return Value

No return value.

Note

Typically, this function must be called from a function OnDeinit(), if the EventSetTimer() function has been called from OnInit(). This function can also be called form the class destructor, if the EventSetTimer() function has been called in the constructor of this class.

Every Expert Advisor, as well as every indicator works with its own timer and receives events only from it. As soon as a mql4 program stops operating, the timer is destroyed forcibly if it was created but hasn't been disabled by the EventKillTimer() function.

EventChartCustom

The function generates a custom event for the specified chart.

```
bool  EventChartCustom(    long     chart_id,            // identifier of t
   ushort  custom_event_id,     // event identifier
   long    lparam,              // parameter of type long
   double  dparam,              // parameter of type double
   string  sparam               // string parameter of the event
   );
```

Parameters

chart_id

[in] Chart identifier. 0 means the current chart.

custom_event_id

[in] ID of the user events. This identifier is automatically added to the value CHARTEVENT_CUSTOM and converted to the integer type.

lparam

[in] Event parameter of the long type passed to the OnChartEvent function.

dparam

[in] Event parameter of the double type passed to the OnChartEvent function.

sparam

[in] Event parameter of the string type passed to the OnChartEvent function. If the string is longer than 63 characters, it is truncated.

Return Value

Returns true if a custom event has been successfully placed in the events queue of the chart that receives the events. In case of failure, it returns false. Use GetLastError() to get an error code.

Note

An Expert Advisor or indicator attached to the specified chart handles the event using the function OnChartEvent(int event_id, long& lparam, double& dparam, string& sparam).

For each type of event, the input parameters of the OnChartEvent() function have definite values that are required for the processing of this event. The events and values passed through this parameters are listed in the below table.

Event	Value of the id parameter	Value of the lpar parameter
Event of a keystroke	CHARTEVENT_KEYDOWN	code of a pressed key
Mouse event (if property CHART_EVENT_MOUSE_MOVE=true is set for the chart)	CHARTEVENT_MOUSE_MOVE	the X coordinate
Event of graphical object creation (if CHART_EVENT_OBJECT_CREATE=true is set for the chart)	CHARTEVENT_OBJECT_CREATE	
Event of change of an object property via the properties dialog	CHARTEVENT_OBJECT_CHANGE	
Event of graphical object deletion (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)	CHARTEVENT_OBJECT_DELETE	
Event of a mouse click on the chart	CHARTEVENT_CLICK	the X coordinate
Event of a mouse click in a graphical object belonging to the chart	CHARTEVENT_OBJECT_CLICK	the X coordinate
Event of a graphical object dragging using the mouse	CHARTEVENT_OBJECT_DRAG	
Event of the finished text editing in the entry box of the LabelEdit graphical object	CHARTEVENT_OBJECT_ENDEDIT	
Event of changes on a chart	CHARTEVENT_CHART_CHANGE	
ID of the user event under the N number	CHARTEVENT_CUSTOM+N	Value set by the EventChartCusto function

Example:

```
//+------------------------------------------------------------------+
//|                                            ButtonClickExpert.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "https://www.mql5.com"
#property version   "1.00"

string buttonID="Button";
string labelID="Info";
int broadcastEventID=5000;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- Create a button to send custom events
   ObjectCreate(0,buttonID,OBJ_BUTTON,0,100,100);
   ObjectSetInteger(0,buttonID,OBJPROP_COLOR,clrWhite);
   ObjectSetInteger(0,buttonID,OBJPROP_BGCOLOR,clrGray);
   ObjectSetInteger(0,buttonID,OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,buttonID,OBJPROP_YDISTANCE,100);
   ObjectSetInteger(0,buttonID,OBJPROP_XSIZE,200);
   ObjectSetInteger(0,buttonID,OBJPROP_YSIZE,50);
   ObjectSetString(0,buttonID,OBJPROP_FONT,"Arial");
   ObjectSetString(0,buttonID,OBJPROP_TEXT,"Button");
   ObjectSetInteger(0,buttonID,OBJPROP_FONTSIZE,10);
   ObjectSetInteger(0,buttonID,OBJPROP_SELECTABLE,0);

//--- Create a label for displaying information
   ObjectCreate(0,labelID,OBJ_LABEL,0,100,100);
   ObjectSetInteger(0,labelID,OBJPROP_COLOR,clrRed);
   ObjectSetInteger(0,labelID,OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,labelID,OBJPROP_YDISTANCE,50);
   ObjectSetString(0,labelID,OBJPROP_FONT,"Trebuchet MS");
   ObjectSetString(0,labelID,OBJPROP_TEXT,"No information");
   ObjectSetInteger(0,labelID,OBJPROP_FONTSIZE,20);
   ObjectSetInteger(0,labelID,OBJPROP_SELECTABLE,0);

//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   ObjectDelete(0,buttonID);
   ObjectDelete(0,labelID);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- Check the event by pressing a mouse button
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      string clickedChartObject=sparam;
      //--- If you click on the object with the name buttonID
      if(clickedChartObject==buttonID)
        {
         //--- State of the button - pressed or not
         bool selected=ObjectGetInteger(0,buttonID,OBJPROP_STATE);
         //--- log a debug message
         Print("Button pressed = ",selected);
         int customEventID; // Number of the custom event to send
         string message;    // Message to be sent in the event
         //--- If the button is pressed
         if(selected)
           {
            message="Button pressed";
            customEventID=CHARTEVENT_CUSTOM+1;
           }
         else // Button is not pressed
           {
            message="Button in not pressed";
            customEventID=CHARTEVENT_CUSTOM+999;
           }
         //--- Send a custom event "our" chart
         EventChartCustom(0,customEventID-CHARTEVENT_CUSTOM,0,0,message);
         ///--- Send a message to all open charts
         BroadcastEvent(ChartID(),0,"Broadcast Message");
         //--- Debug message
         Print("Sent an event with ID = ",customEventID);
        }
      ChartRedraw();// Forced redraw all chart objects
     }

//--- Check the event belongs to the user events
   if(id>CHARTEVENT_CUSTOM)
     {
      if(id==broadcastEventID)
        {
         Print("Got broadcast message from a chart with id = "+lparam);
        }
      else
        {
         //--- We read a text message in the event
         string info=sparam;
         Print("Handle the user event with the ID = ",id);
         //--- Display a message in a label
         ObjectSetString(0,labelID,OBJPROP_TEXT,sparam);
         ChartRedraw();// Forced redraw all chart objects
        }
     }
  }
//+------------------------------------------------------------------+
//| sends broadcast event to all open charts                         |
//+------------------------------------------------------------------+
void BroadcastEvent(long lparam,double dparam,string sparam)
  {
   int eventID=broadcastEventID-CHARTEVENT_CUSTOM;
   long currChart=ChartFirst();
   int i=0;
   while(i<CHARTS_MAX)                 // We have certainly no more than C
     {
      EventChartCustom(currChart,eventID,lparam,dparam,sparam);
      currChart=ChartNext(currChart); // We have received a new chart from
      if(currChart==-1) break;        // Reached the end of the charts lis
      i++;// Do not forget to increase the counter
     }
  }
//+------------------------------------------------------------------+
```

See also

Events of the client terminal, Event handler functions

Obsolete functions

In further development of MQL4, some functions were renamed and moved from one group to another in order to systematize them better.

The old function names are not highlighted or linked to the MetaEditor key names dictionary. The old function names can be used since the compiler will accept them in a proper way. However, we strongly recommend to use the new names.

Old name	New Name
BarsPerWindow	WindowBarsPerChart
ClientTerminalName	TerminalName
CurTime	TimeCurrent
CompanyName	TerminalCompany
FirstVisibleBar	WindowFirstVisibleBar
Highest	iHighest
HistoryTotal	OrdersHistoryTotal
LocalTime	TimeLocal
Lowest	iLowest
ObjectsRedraw	WindowRedraw
PriceOnDropped	WindowPriceOnDropped
ScreenShot	WindowScreenShot
ServerAddress	AccountServer
TimeOnDropped	WindowTimeOnDropped

List of MQL4 Functions

All MQL4 functions in alphabetical order.

Function	Action	Section
AccountBalance	Returns balance value of the current account	Account Information
AccountCompany	Returns the brokerage company name where the current account was registered	Account Information
AccountCredit	Returns credit value of the current account	Account Information
AccountCurrency	Returns currency name of the current account	Account Information
AccountEquity	Returns equity value of the current account	Account Information
AccountFreeMargin	Returns free margin value of the current account	Account Information
AccountFreeMarginCheck	Returns free margin that remains after the specified position has been opened at the current price on the current account	Account Information
AccountFreeMarginMode	Calculation mode of free margin allowed to open orders on the current account	Account Information
AccountInfoDouble	Returns a value of double type of the corresponding account property	Account Information
AccountInfoInteger	Returns a value of integer type (bool, int or long) of the corresponding account property	Account Information
AccountInfoString	Returns a value string type corresponding account property	Account Information
AccountLeverage	Returns leverage of the current account	Account Information
AccountMargin	Returns margin value of the current account	Account Information
AccountName	Returns the current account name	Account Information

AccountNumber	Returns the current account number	Account Information
AccountProfit	Returns profit value of the current account	Account Information
AccountServer	Returns the connected server name	Account Information
AccountStopoutLevel	Returns the value of the Stop Out level	Account Information
AccountStopoutMode	Returns the calculation mode for the Stop Out level	Account Information
acos	Returns the arc cosine of x in radians	Math Functions
Alert	Displays a message in a separate window	Common Functions
ArrayBsearch	Returns index of the first found element in the first array dimension	Array Functions
ArrayCompare	Returns the result of comparing two arrays of simple types or custom structures without complex objects	Array Functions
ArrayCopy	Copies one array into another	Array Functions
ArrayCopyRates	Copies rates to the two-dimensional array from chart RateInfo array returns copied bars amount	Array Functions
ArrayCopySeries	Copies a series array to another one and returns the count of the copied elements	Array Functions
ArrayDimension	Returns the multidimensional array rank	Array Functions
ArrayFill	Fills an array with the specified value	Array Functions
ArrayFree	Frees up buffer of any dynamic array and sets the size of the zero dimension in 0.	Array Functions
ArrayGetAsSeries	Checks direction of array indexing	Array Functions
ArrayInitialize	Sets all elements of a numeric array into a single value	Array Functions

ArrayIsDynamic	Checks whether an array is dynamic	Array Functions
ArrayIsSeries	Checks whether an array is a timeseries	Array Functions
ArrayMaximum	Search for an element with the maximal value	Array Functions
ArrayMinimum	Search for an element with the minimal value	Array Functions
ArrayRange	Returns the number of elements in the specified dimension of the array	Array Functions
ArrayResize	Sets the new size in the first dimension of the array	Array Functions
ArraySetAsSeries	Sets the direction of array indexing	Array Functions
ArraySize	Returns the number of elements in the array	Array Functions
ArraySort	Sorting of numeric arrays by the first dimension	Array Functions
asin	Returns the arc sine of x in radians	Math Functions
atan	Returns the arc tangent of x in radians	Math Functions
Bars	Returns the number of bars count in the history for a specified symbol and period	Timeseries and Indicators Access
ceil	Returns integer numeric value closest from above	Math Functions
CharArrayToString	Converting symbol code (ansi) into one-symbol array	Conversion Functions
ChartApplyTemplate	Applies a specific template from a specified file to the chart	Chart Operations
ChartClose	Closes the specified chart	Chart Operations
ChartFirst	Returns the ID of the first chart of the client terminal	Chart Operations
ChartGetDouble	Returns the double value property of the	Chart

	specified chart	Operations
ChartGetInteger	Returns the integer value property of the specified chart	Chart Operations
ChartGetString	Returns the string value property of the specified chart	Chart Operations
ChartID	Returns the ID of the current chart	Chart Operations
ChartIndicatorDelete	Removes an indicator with a specified name from the specified chart window	Chart Operations
ChartIndicatorName	Returns the short name of the indicator by the number in the indicators list on the specified chart window	Chart Operations
ChartIndicatorsTotal	Returns the number of all indicators applied to the specified chart window.	Chart Operations
ChartNavigate	Performs shift of the specified chart by the specified number of bars relative to the specified position in the chart	Chart Operations
ChartNext	Returns the chart ID of the chart next to the specified one	Chart Operations
ChartOpen	Opens a new chart with the specified symbol and period	Chart Operations
CharToStr	Conversion of the symbol code into a one-character string	Conversion Functions
CharToString	Converting a symbol code into a one-character string	Conversion Functions
ChartPeriod	Returns the period value of the specified chart	Chart Operations
ChartPriceOnDropped	Returns the price coordinate of the chart point, the Expert Advisor or script has been dropped to	Chart Operations
ChartRedraw	Calls a forced redrawing of a specified chart	Chart Operations
ChartSaveTemplate	Saves current chart settings in a template with a specified name	Chart Operations
ChartScreenShot	Provides a screenshot of the chart of its current state in a gif format	Chart Operations
ChartSetDouble	Sets the double value for a corresponding	Chart

	property of the specified chart	Operations
ChartSetInteger	Sets the integer value (datetime, int, color, bool or char) for a corresponding property of the specified chart	Chart Operations
ChartSetString	Sets the string value for a corresponding property of the specified chart	Chart Operations
ChartSetSymbolPeriod	Changes the symbol value and a period of the specified chart	Chart Operations
ChartSymbol	Returns the symbol name of the specified chart	Chart Operations
ChartTimeOnDropped	Returns the time coordinate of the chart point, the Expert Advisor or script has been dropped to	Chart Operations
ChartTimePriceToXY	Converts the coordinates of a chart from the time/price representation to the X and Y coordinates	Chart Operations
ChartWindowFind	Returns the number of a subwindow where an indicator is drawn	Chart Operations
ChartWindowOnDropped	Returns the number (index) of the chart subwindow, the Expert Advisor or script has been dropped to	Chart Operations
ChartXOnDropped	Returns the X coordinate of the chart point, the Expert Advisor or script has been dropped to	Chart Operations
ChartXYToTimePrice	Converts the X and Y coordinates on a chart to the time and price values	Chart Operations
ChartYOnDropped	Returns the Y coordinate of the chart point, the Expert Advisor or script has been dropped to	Chart Operations
CheckPointer	Returns the type of the object pointer	Common Functions
ColorToARGB	Converting color type to uint type to receive ARGB representation of the color.	Conversion Functions
ColorToString	Converting color value into string as "R,G,B"	Conversion Functions
Comment	Outputs a comment in the left top corner of the chart	Common Functions
CopyClose	Gets history data on bar closing price for	Timeseries

	a specified symbol and period into an array	and Indicators Access
CopyHigh	Gets history data on maximal bar price for a specified symbol and period into an array	Timeseries and Indicators Access
CopyLow	Gets history data on minimal bar price for a specified symbol and period into an array	Timeseries and Indicators Access
CopyOpen	Gets history data on bar opening price for a specified symbol and period into an array	Timeseries and Indicators Access
CopyRates	Gets history data of the Rates structure for a specified symbol and period into an array	Timeseries and Indicators Access
CopyTickVolume	Gets history data on tick volumes for a specified symbol and period into an array	Timeseries and Indicators Access
CopyTime	Gets history data on bar opening time for a specified symbol and period into an array	Timeseries and Indicators Access
cos	Returns the cosine of a number	Math Functions
CryptDecode	Performs the inverse transformation of the data from array	Common Functions
CryptEncode	Transforms the data from array with the specified method	Common Functions
Day	Returns the current day of the month, i.e., the day of month of the last known server time	Date and Time
DayOfWeek	Returns the current zero-based day of the week of the last known server time	Date and Time
DayOfYear	Returns the current day of the year i.e.,	Date and

	the day of year of the last known server time	Time
DebugBreak	Program breakpoint in debugging	Common Functions
Digits	Returns the number of decimal digits determining the accuracy of the price value of the current chart symbol	Checkup
DoubleToStr	Returns text string with the specified numerical value converted into a specified precision format	Conversion Functions
DoubleToString	Converting a numeric value to a text line with a specified accuracy	Conversion Functions
EnumToString	Converting an enumeration value of any type to string	Conversion Functions
EventChartCustom	Generates a custom event for the specified chart	Working with Events
EventKillTimer	Stops the generation of events by the timer in the current chart	Working with Events
EventSetMillisecondTimer	Launches event generator of the high-resolution timer with a period less than 1 second for the current chart	Working with Events
EventSetTimer	Starts the timer event generator with the specified periodicity for the current chart	Working with Events
exp	Returns exponent of a number	Math Functions
ExpertRemove	Stops Expert Advisor and unloads it from the chart	Common Functions
fabs	Returns absolute value (modulus) of the specified numeric value	Math Functions
FileClose	Closes a previously opened file	File Functions
FileCopy	Copies the original file from a local or shared folder to another file	File Functions
FileDelete	Deletes a specified file	File Functions
FileFindClose	Closes search handle	File

		Functions
FileFindFirst	Starts the search of files in a directory in accordance with the specified filter	File Functions
FileFindNext	Continues the search started by the FileFindFirst() function	File Functions
FileFlush	Writes to a disk all data remaining in the input/output file buffer	File Functions
FileGetInteger	Gets an integer property of a file	File Functions
FileIsEnding	Defines the end of a file in the process of reading	File Functions
FileIsExist	Checks the existence of a file	File Functions
FileIsLineEnding	Defines the end of a line in a text file in the process of reading	File Functions
FileMove	Moves or renames a file	File Functions
FileOpen	Opens a file with a specified name and flag	File Functions
FileOpenHistory	Opens file in the current history directory or in its subfolders	File Functions
FileReadArray	Reads arrays of any type except for string from the file of the BIN type	File Functions
FileReadBool	Reads from the file of the CSV type a string from the current position till a delimiter (or till the end of a text line) and converts the read string to a value of bool type	File Functions
FileReadDatetime	Reads from the file of the CSV type a string of one of the formats: "YYYY.MM.DD HH:MM:SS", "YYYY.MM.DD" or "HH:MM:SS" - and converts it into a datetime value	File Functions
FileReadDouble	Reads a double value from the current position of the file pointer	File Functions
FileReadFloat	Reads a float value from the current position of the file pointer	File Functions
FileReadInteger	Reads int, short or char value from the	File

	current position of the file pointer	Functions
FileReadLong	Reads a long type value from the current position of the file pointer	File Functions
FileReadNumber	Reads from the file of the CSV type a string from the current position till a delimiter (or til the end of a text line) and converts the read string into double value	File Functions
FileReadString	Reads a string from the current position of a file pointer from a file	File Functions
FileReadStruct	Reads the contents from a binary file into a structure passed as a parameter, from the current position of the file pointer	File Functions
FileSeek	Moves the position of the file pointer by a specified number of bytes relative to the specified position	File Functions
FileSize	Returns the size of a corresponding open file	File Functions
FileTell	Returns the current position of the file pointer of a corresponding open file	File Functions
FileWrite	Writes data to a file of CSV or TXT type	File Functions
FileWriteArray	Writes arrays of any type except for string into a file of BIN type	File Functions
FileWriteDouble	Writes value of the double type from the current position of a file pointer into a binary file	File Functions
FileWriteFloat	Writes value of the float type from the current position of a file pointer into a binary file	File Functions
FileWriteInteger	Writes value of the int type from the current position of a file pointer into a binary file	File Functions
FileWriteLong	Writes value of the long type from the current position of a file pointer into a binary file	File Functions
FileWriteString	Writes the value of a string parameter	File

	into a BIN or TXT file starting from the current position of the file pointer	Functions
FileWriteStruct	Writes the contents of a structure passed as a parameter into a binary file, starting from the current position of the file pointer	File Functions
floor	Returns integer numeric value closest from below	Math Functions
fmax	Returns the maximal value of the two numeric values	Math Functions
fmin	Returns the minimal value of the two numeric values	Math Functions
fmod	Returns the real remainder after the division of two numbers	Math Functions
FolderClean	Deletes all files in the specified folder	File Functions
FolderCreate	Creates a folder in the Files directory	File Functions
FolderDelete	Removes a selected directory. If the folder is not empty, then it can't be removed	File Functions
GetLastError	Returns the last error	Checkup
GetPointer	Returns the object pointer	Common Functions
GetTickCount	Returns the number of milliseconds that have elapsed since the system was started	Common Functions
GlobalVariableCheck	Checks the existence of a global variable with the specified name	Global Variables of the Terminal
GlobalVariableDel	Deletes a global variable	Global Variables of the Terminal
GlobalVariableGet	Returns the value of a global variable	Global Variables of the

		Terminal
GlobalVariableName	Returns the name of a global variable by it's ordinal number in the list of global variables	Global Variables of the Terminal
GlobalVariablesDeleteAll	Deletes global variables with the specified prefix in their names	Global Variables of the Terminal
GlobalVariableSet	Sets the new value to a global variable	Global Variables of the Terminal
GlobalVariableSetOnCondition	Sets the new value of the existing global variable by condition	Global Variables of the Terminal
GlobalVariablesFlush	Forcibly saves contents of all global variables to a disk	Global Variables of the Terminal
GlobalVariablesTotal	Returns the total number of global variables	Global Variables of the Terminal
GlobalVariableTemp	Sets the new value to a global variable, that exists only in the current session of the terminal	Global Variables of the Terminal
GlobalVariableTime	Returns time of the last accessing the global variable	Global Variables of the Terminal
HideTestIndicators	The function sets a flag hiding indicators called by the Expert Advisor	Custom Indicators
Hour	Returns the hour of the last known server time by the moment of the program start	Date and Time
iAC	Accelerator Oscillator	Technical Indicators

iAD	Accumulation/Distribution	Technical Indicators
iADX	Average Directional Index	Technical Indicators
iAlligator	Alligator	Technical Indicators
iAO	Awesome Oscillator	Technical Indicators
iATR	Average True Range	Technical Indicators
iBands	Bollinger Bands®	Technical Indicators
iBandsOnArray	Calculation of Bollinger Bands® indicator on data, stored in a numeric array	Technical Indicators
iBars	Returns the number of bars on the specified chart	Timeseries and Indicators Access
iBarShift	Returns the index of the bar which covers the specified time	Timeseries and Indicators Access
iBearsPower	Bears Power	Technical Indicators
iBullsPower	Bulls Power	Technical Indicators
iBWMFI	Market Facilitation Index by Bill Williams	Technical Indicators
iCCI	Commodity Channel Index	Technical Indicators
iCCIOnArray	Calculation of Commodity Channel Index indicator on data, stored in a numeric array	Technical Indicators
iClose	Returns Close price value for the bar of specified symbol with timeframe and shift	Timeseries and Indicators Access
iCustom	Custom indicator	Technical

		Indicators
iDeMarker	DeMarker	Technical Indicators
iEnvelopes	Envelopes	Technical Indicators
iEnvelopesOnArray	Calculation of Envelopes indicator on data, stored in a numeric array	Technical Indicators
iForce	Force Index	Technical Indicators
iFractals	Fractals	Technical Indicators
iGator	Gator Oscillator	Technical Indicators
iHigh	Returns High price value for the bar of specified symbol with timeframe and shift	Timeseries and Indicators Access
iHighest	Returns the shift of the maximum value over a specific number of bars	Timeseries and Indicators Access
iIchimoku	Ichimoku Kinko Hyo	Technical Indicators
iLow	Returns Low price value for the bar of indicated symbol with timeframe and shift	Timeseries and Indicators Access
iLowest	Returns the shift of the lowest value over a specific number of bars	Timeseries and Indicators Access
iMA	Moving Average	Technical Indicators
iMACD	Moving Averages Convergence-Divergence	Technical Indicators
iMAOnArray	Calculation of Moving Average indicator on data, stored in a numeric array	Technical Indicators
iMFI	Money Flow Index	Technical

		Indicators
iMomentum	Momentum	Technical Indicators
iMomentumOnArray	Calculation of Momentum indicator on data, stored in a numeric array	Technical Indicators
IndicatorBuffers	Allocates memory for buffers used for custom indicator calculations	Custom Indicators
IndicatorCounted	Returns the amount of bars not changed after the indicator had been launched last	Custom Indicators
IndicatorDigits	Sets precision format to visualize indicator values	Custom Indicators
IndicatorSetDouble	Sets the value of an indicator property of the double type	Custom Indicators
IndicatorSetInteger	Sets the value of an indicator property of the int type	Custom Indicators
IndicatorSetString	Sets the value of an indicator property of the string type	Custom Indicators
IndicatorShortName	Sets the "short" name of a custom indicator to be shown in the DataWindow and in the chart subwindow	Custom Indicators
IntegerToString	Converting int into a string of preset length	Conversion Functions
iOBV	On Balance Volume	Technical Indicators
iOpen	Returns Open price value for the bar of specified symbol with timeframe and shift	Timeseries and Indicators Access
iOsMA	Moving Average of Oscillator (MACD histogram)	Technical Indicators
iRSI	Relative Strength Index	Technical Indicators
iRSIOnArray	Calculation of Momentum indicator on data, stored in a numeric array	Technical Indicators
iRVI	Relative Vigor Index	Technical Indicators

iSAR	Parabolic Stop And Reverse System	Technical Indicators
IsConnected	Checks connection between client terminal and server	Checkup
IsDemo	Checks if the Expert Advisor runs on a demo account	Checkup
IsDllsAllowed	Checks if the DLL function call is allowed for the Expert Advisor	Checkup
IsExpertEnabled	Checks if Expert Advisors are enabled for running	Checkup
IsLibrariesAllowed	Checks if the Expert Advisor can call library function	Checkup
IsOptimization	Checks if Expert Advisor runs in the Strategy Tester optimization mode	Checkup
IsStopped	Returns true, if an mql4 program has been commanded to stop its operation	Checkup
iStdDev	Standard Deviation	Technical Indicators
iStdDevOnArray	Calculation of Standard Deviation indicator on data, stored in a numeric array	Technical Indicators
IsTesting	Checks if the Expert Advisor runs in the testing mode	Checkup
iStochastic	Stochastic Oscillator	Technical Indicators
IsTradeAllowed	Checks if the Expert Advisor is allowed to trade and trading context is not busy	Checkup
IsTradeContextBusy	Returns the information about trade context	Checkup
IsVisualMode	Checks if the Expert Advisor is tested in visual mode	Checkup
iTime	Returns time value for the bar of specified symbol with timeframe and shift	Timeseries and Indicators Access
iVolume	Returns Tick Volume value for the bar of specified symbol with timeframe and	Timeseries and

	shift	Indicators Access
iWPR	Williams' Percent Range	Technical Indicators
log	Returns natural logarithm	Math Functions
log10	Returns the logarithm of a number by base 10	Math Functions
MarketInfo	Returns various data about securities listed in the "Market Watch" window	Market Info
MathAbs	Returns absolute value (modulus) of the specified numeric value	Math Functions
MathArccos	Returns the arc cosine of x in radians	Math Functions
MathArcsin	Returns the arc sine of x in radians	Math Functions
MathArctan	Returns the arc tangent of x in radians	Math Functions
MathCeil	Returns integer numeric value closest from above	Math Functions
MathCos	Returns the cosine of a number	Math Functions
MathExp	Returns exponent of a number	Math Functions
MathFloor	Returns integer numeric value closest from below	Math Functions
MathIsValidNumber	Checks the correctness of a real number	Math Functions
MathLog	Returns natural logarithm	Math Functions
MathLog10	Returns the logarithm of a number by base 10	Math Functions
MathMax	Returns the maximal value of the two numeric values	Math Functions
MathMin	Returns the minimal value of the two numeric values	Math Functions

MathMod	Returns the real remainder after the division of two numbers	Math Functions
MathPow	Raises the base to the specified power	Math Functions
MathRand	Returns a pseudorandom value within the range of 0 to 32767	Math Functions
MathRound	Rounds of a value to the nearest integer	Math Functions
MathSin	Returns the sine of a number	Math Functions
MathSqrt	Returns a square root	Math Functions
MathSrand	Sets the starting point for generating a series of pseudorandom integers	Math Functions
MathTan	Returns the tangent of a number	Math Functions
MessageBox	Creates, displays a message box and manages it	Common Functions
Minute	Returns the current minute of the last known server time by the moment of the program start	Date and Time
Month	Returns the current month as number, i.e., the number of month of the last known server time	Date and Time
MQLInfoInteger	Returns an integer value of a corresponding property of a running mql4 program	Checkup
MQLInfoString	Returns a string value of a corresponding property of a running mql4 program	Checkup
MQLSetInteger	Sets the value of the MQL_CODEPAGE property in an MQL4 program environment	Checkup
NormalizeDouble	Rounding of a floating point number to a specified accuracy	Conversion Functions
ObjectCreate	Creates an object of the specified type in a specified chart	Object Functions
ObjectDelete	Removes the object having the specified	Object

	name	Functions
ObjectDescription	Returns the object description	Object Functions
ObjectFind	Searches for an object having the specified name	Object Functions
ObjectGet	Returns the value of the specified object property	Object Functions
ObjectGetDouble	Returns the double value of the corresponding object property	Object Functions
ObjectGetFiboDescription	Returns the level description of a Fibonacci object	Object Functions
ObjectGetInteger	Returns the integer value of the corresponding object property	Object Functions
ObjectGetShiftByValue	Calculates and returns bar index for the given price	Object Functions
ObjectGetString	Returns the string value of the corresponding object property	Object Functions
ObjectGetTimeByValue	Returns the time value for the specified object price value	Object Functions
ObjectGetValueByShift	Calculates and returns the price value for the specified bar	Object Functions
ObjectGetValueByTime	Returns the price value of an object for the specified time	Object Functions
ObjectMove	Changes the coordinates of the specified object anchor point	Object Functions
ObjectName	Returns the name of an object by its index in the objects list	Object Functions
ObjectsDeleteAll	Removes all objects of the specified type from the specified chart subwindow	Object Functions
ObjectSet	Changes the value of the specified object property	Object Functions
ObjectSetDouble	Sets the value of the corresponding object property	Object Functions
ObjectSetFiboDescription	Sets a new description to a level of a Fibonacci object	Object Functions
ObjectSetInteger	Sets the value of the corresponding object property	Object Functions

ObjectSetString	Sets the value of the corresponding object property	Object Functions
ObjectSetText	Changes the object description	Object Functions
ObjectsTotal	Returns the number of objects of the specified type	Object Functions
ObjectType	Returns the object type	Object Functions
OrderClose	Closes opened order	Trade Functions
OrderCloseBy	Closes an opened order by another opposite opened order	Trade Functions
OrderClosePrice	Returns close price of the currently selected order	Trade Functions
OrderCloseTime	Returns close time of the currently selected order	Trade Functions
OrderComment	Returns comment of the currently selected order	Trade Functions
OrderCommission	Returns calculated commission of the currently selected order	Trade Functions
OrderDelete	Deletes previously opened pending order	Trade Functions
OrderExpiration	Returns expiration date of the selected pending order	Trade Functions
OrderLots	Returns amount of lots of the selected order	Trade Functions
OrderMagicNumber	Returns an identifying (magic) number of the currently selected order	Trade Functions
OrderModify	Modification of characteristics of the previously opened or pending orders	Trade Functions
OrderOpenPrice	Returns open price of the currently selected order	Trade Functions
OrderOpenTime	Returns open time of the currently selected order	Trade Functions
OrderPrint	Prints information about the selected order in the log	Trade Functions

OrderProfit	Returns profit of the currently selected order	Trade Functions
OrderSelect	The function selects an order for further processing	Trade Functions
OrderSend	The main function used to open an order or place a pending order	Trade Functions
OrdersHistoryTotal	Returns the number of closed orders in the account history loaded into the terminal	Trade Functions
OrderStopLoss	Returns stop loss value of the currently selected order	Trade Functions
OrdersTotal	Returns the number of market and pending orders	Trade Functions
OrderSwap	Returns swap value of the currently selected order	Trade Functions
OrderSymbol	Returns symbol name of the currently selected order	Trade Functions
OrderTakeProfit	Returns take profit value of the currently selected order	Trade Functions
OrderTicket	Returns ticket number of the currently selected order	Trade Functions
OrderType	Returns order operation type of the currently selected order	Trade Functions
Period	Returns the current chart timeframe	Checkup
Period	Returns timeframe of the current chart	Chart Operations
PeriodSeconds	Returns the number of seconds in the period	Common Functions
PlaySound	Plays a sound file	Common Functions
Point	Returns the point size of the current symbol in the quote currency	Checkup
pow	Raises the base to the specified power	Math Functions
Print	Displays a message in the log	Common Functions

PrintFormat	Formats and prints the sets of symbols and values in a log file in accordance with a preset format	Common Functions
rand	Returns a pseudorandom value within the range of 0 to 32767	Math Functions
RefreshRates	Refreshing of data in pre-defined variables and series arrays	Timeseries and Indicators Access
ResetLastError	Sets the value of a predetermined variable _LastError to zero	Common Functions
ResourceCreate	Creates an image resource based on a data set	Common Functions
ResourceFree	Deletes dynamically created resource (freeing the memory allocated for it)	Common Functions
ResourceReadImage	Reads data from the graphical resource created by ResourceCreate() function or saved in EX4 file during compilation	Common Functions
ResourceSave	Saves a resource into the specified file	Common Functions
round	Rounds of a value to the nearest integer	Math Functions
Seconds	Returns the amount of seconds elapsed from the beginning of the current minute of the last known server time by the moment of the program start	Date and Time
SendFTP	Sends a file at the address specified in the settings window of the "FTP" tab	Common Functions
SendMail	Sends an email at the address specified in the settings window of the "Email" tab	Common Functions
SendNotification	Sends push notifications to mobile terminals, whose MetaQuotes ID are specified in the "Notifications" tab	Common Functions
SeriesInfoInteger	Returns information about the state of historical data	Timeseries and Indicators Access
SetIndexArrow	Sets an arrow symbol for indicators line	Custom

	of the DRAW_ARROW type	Indicators
SetIndexBuffer	Binds the specified indicator buffer with one-dimensional dynamic array of the double type	Custom Indicators
SetIndexDrawBegin	Sets the bar number from which the drawing of the given indicator line must start	Custom Indicators
SetIndexEmptyValue	Sets drawing line empty value	Custom Indicators
SetIndexLabel	Sets drawing line description for showing in the DataWindow and in the tooltip	Custom Indicators
SetIndexShift	Sets offset for the drawing line	Custom Indicators
SetIndexStyle	Sets the new type, style, width and color for a given indicator line	Custom Indicators
SetLevelStyle	Sets a new style, width and color of horizontal levels of indicator to be output in a separate window	Custom Indicators
SetLevelValue	Sets a value for a given horizontal level of the indicator to be output in a separate window	Custom Indicators
ShortArrayToString	Copying array part into a string	Conversion Functions
ShortToString	Converting symbol code (unicode) into one-symbol string	Conversion Functions
SignalBaseGetDouble	Returns the value of double type property for selected signal	Trade Signals
SignalBaseGetInteger	Returns the value of integer type property for selected signal	Trade Signals
SignalBaseGetString	Returns the value of string type property for selected signal	Trade Signals
SignalBaseSelect	Selects a signal from signals, available in terminal for further working with it	Trade Signals
SignalBaseTotal	Returns the total amount of signals, available in terminal	Trade Signals
SignalInfoGetDouble	Returns the value of double type property of signal copy settings	Trade Signals

SignalInfoGetInteger	Returns the value of integer type property of signal copy settings	Trade Signals
SignalInfoGetString	Returns the value of string type property of signal copy settings	Trade Signals
SignalInfoSetDouble	Sets the value of double type property of signal copy settings	Trade Signals
SignalInfoSetInteger	Sets the value of integer type property of signal copy settings	Trade Signals
SignalSubscribe	Subscribes to the trading signal	Trade Signals
SignalUnsubscribe	Cancels subscription	Trade Signals
sin	Returns the sine of a number	Math Functions
Sleep	Suspends execution of the current Expert Advisor or script within a specified interval	Common Functions
sqrt	Returns a square root	Math Functions
srand	Sets the starting point for generating a series of pseudorandom integers	Math Functions
StringAdd	Adds a string to the end of another string	String Functions
StringBufferLen	Returns the size of buffer allocated for the string	String Functions
StringCompare	Compares two strings and returns 1 if the first string is greater than the second; 0 - if the strings are equal; -1 (minus 1) - if the first string is less than the second one	String Functions
StringConcatenate	Forms a string of parameters passed	String Functions
StringFill	Fills out a specified string by selected symbols	String Functions
StringFind	Search for a substring in a string	String Functions
StringFormat	Converting number into string according	Conversion

	to preset format	Functions
StringGetChar	Returns character (code) from the specified position in the string	String Functions
StringGetCharacter	Returns the value of a number located in the specified string position	String Functions
StringInit	Initializes string by specified symbols and provides the specified string length	String Functions
StringLen	Returns the number of symbols in a string	String Functions
StringReplace	Replaces all the found substrings of a string by a set sequence of symbols	String Functions
StringSetChar	Returns the string copy with changed character in the specified position	String Functions
StringSetCharacter	Returns true is a symbol is successfully inserted to the passed string.	String Functions
StringSplit	Gets substrings by a specified separator from the specified string, returns the number of substrings obtained	String Functions
StringSubstr	Extracts a substring from a text string starting from a specified position	String Functions
StringToCharArray	Symbol-wise copying a string converted from Unicode to ANSI, to a selected place of array of uchar type	Conversion Functions
StringToColor	Converting "R,G,B" string or string with color name into color type value	Conversion Functions
StringToDouble	Converting a string containing a symbol representation of number into number of double type	Conversion Functions
StringToInteger	Converting a string containing a symbol representation of number into number of int type	Conversion Functions
StringToLower	Transforms all symbols of a selected string to lowercase by location	String Functions
StringToShortArray	Symbol-wise copying a string to a selected part of array of ushort type	Conversion Functions
StringToTime	Converting a string containing time or date in "yyyy.mm.dd [hh:mi]" format into	Conversion Functions

	datetime type	
StringToUpper	Transforms all symbols of a selected string into capitals by location	String Functions
StringTrimLeft	Cuts line feed characters, spaces and tabs in the left part of the string	String Functions
StringTrimRight	Cuts line feed characters, spaces and tabs in the right part of the string	String Functions
StrToDouble	Converts string representation of number to double type	Conversion Functions
StrToInteger	Converts string containing the value character representation into a value of the integer type	Conversion Functions
StrToTime	Converts string in the format "yyyy.mm.dd hh:mi" to datetime type	Conversion Functions
StructToTime	Converts a variable of MqlDateTime structure type into a datetime value	Date and Time
Symbol	Returns the name of a symbol of the current chart	Checkup
Symbol	Returns a text string with the name of the current financial instrument	Chart Operations
SymbolInfoDouble	Returns the double value of the symbol for the corresponding property	Market Info
SymbolInfoInteger	Returns a value of an integer type (long, datetime, int or bool) of a specified symbol for the corresponding property	Market Info
SymbolInfoSessionQuote	Allows receiving time of beginning and end of the specified quoting sessions for a specified symbol and day of week.	Market Info
SymbolInfoSessionTrade	Allows receiving time of beginning and end of the specified trading sessions for a specified symbol and day of week.	Market Info
SymbolInfoString	Returns a value of the string type of a specified symbol for the corresponding property	Market Info
SymbolInfoTick	Returns the current prices for the specified symbol in a variable of the MqlTick type	Market Info
SymbolName	Returns the name of a specified symbol	Market Info

SymbolSelect	Selects a symbol in the Market Watch window or removes a symbol from the window	Market Info
SymbolsTotal	Returns the number of available (selected in Market Watch or all) symbols	Market Info
tan	Returns the tangent of a number	Math Functions
TerminalClose	Commands the terminal to complete operation	Common Functions
TerminalCompany	Returns the name of company owning the client terminal	Checkup
TerminalInfoDouble	Returns an double value of a corresponding property of a running mql4 program	Checkup
TerminalInfoInteger	Returns an integer value of a corresponding property of a running mql4 program	Checkup
TerminalInfoString	Returns a string value of a corresponding property of a running mql4 program	Checkup
TerminalName	Returns client terminal name	Checkup
TerminalPath	Returns the directory, from which the client terminal was launched	Checkup
TesterStatistics	It returns the value of a specified statistic calculated based on testing results	Common Functions
TextGetSize	Returns the string's width and height at the current font settings	Object Functions
TextOut	Transfers the text to the custom array (buffer) designed for creation of a graphical resource	Object Functions
TextSetFont	Sets the font for displaying the text using drawing methods (Arial 20 used by default)	Object Functions
TimeCurrent	Returns the last known server time (time of the last quote receipt) in the datetime format	Date and Time
TimeDay	Returns the day of month of the specified	Date and

	date	Time
TimeDaylightSavings	Returns the sign of Daylight Saving Time switch	Date and Time
TimeDayOfWeek	Returns the zero-based day of week of the specified date	Date and Time
TimeDayOfYear	Returns the day of year of the specified date	Date and Time
TimeGMT	Returns GMT in datetime format with the Daylight Saving Time by local time of the computer, where the client terminal is running	Date and Time
TimeGMTOffset	Returns the current difference between GMT time and the local computer time in seconds, taking into account DST switch	Date and Time
TimeHour	Returns the hour of the specified time	Date and Time
TimeLocal	Returns the local computer time in datetime format	Date and Time
TimeMinute	Returns the minute of the specified time	Date and Time
TimeMonth	Returns the month number of the specified time	Date and Time
TimeSeconds	Returns the amount of seconds elapsed from the beginning of the minute of the specified time	Date and Time
TimeToStr	Converts value of datetime type into a string of "yyyy.mm.dd hh:mi" format	Conversion Functions
TimeToString	Converting a value containing time in seconds elapsed since 01.01.1970 into a string of "yyyy.mm.dd hh:mi" format	Conversion Functions
TimeToStruct	Converts a datetime value into a variable of MqlDateTime structure type	Date and Time
TimeYear	Returns year of the specified date	Date and Time
UninitializeReason	Returns the code of the reason for deinitialization	Checkup
WebRequest	Sends HTTP request to the specified	Common

	server	Functions
WindowBarsPerChart	Returns the amount of bars visible on the chart	Chart Operations
WindowExpertName	Returns the name of the executed Expert Advisor, script, custom indicator, or library	Chart Operations
WindowFind	Returns the window index containing this specified indicator	Chart Operations
WindowFirstVisibleBar	Returns index of the first visible bar in the current chart window	Chart Operations
WindowHandle	Returns the system handle of the chart window	Chart Operations
WindowIsVisible	Returns the visibility flag of the chart subwindow	Chart Operations
WindowOnDropped	Returns the window index where Expert Advisor, custom indicator or script was dropped	Chart Operations
WindowPriceMax	Returns the maximal value of the vertical scale of the specified subwindow of the current chart	Chart Operations
WindowPriceMin	Returns the minimal value of the vertical scale of the specified subwindow of the current chart	Chart Operations
WindowPriceOnDropped	Returns the price of the chart point where Expert Advisor or script was dropped	Chart Operations
WindowRedraw	Redraws the current chart forcedly	Chart Operations
WindowScreenShot	Saves current chart screen shot as a GIF, PNG or BMP file depending on specified extension	Chart Operations
WindowsTotal	Returns total number of indicator windows on the chart	Chart Operations
WindowTimeOnDropped	Returns the time of the chart point where Expert Advisor or script was dropped	Chart Operations
WindowXOnDropped	Returns the value at X axis in pixels for the chart window client area point at	Chart Operations

	which the Expert Advisor or script was dropped	
WindowYOnDropped	Returns the value at Y axis in pixels for the chart window client area point at which the Expert Advisor or script was dropped	Chart Operations
Year	Returns the current year, i.e., the year of the last known server time	Date and Time
ZeroMemory	Resets a variable passed to it by reference. The variable can be of any type, except for classes and structures that have constructors.	Common Functions

List of MQL4 Constants

All MQL4 constants in alphabetical order.

Constant	Description	Us
__DATE__	File compilation date without time (hours, minutes and seconds are equal to 0)	Pr Su
__DATETIME__	File compilation date and time	Pr Su
__FILE__	Name of the currently compiled file	Pr Su
__FUNCSIG__	Signature of the function in whose body the macro is located. Logging of the full description of functions can be useful in the identification of overloaded functions	Pr Su
__FUNCTION__	Name of the function, in whose body the macro is located	Pr Su
__LINE__	Line number in the source code, in which the macro is located	Pr Su
__MQLBUILD__, __MQL4BUILD__	Compiler build number	Pr Su
__PATH__	An absolute path to the file that is currently being compiled	Pr Su
ACCOUNT_BALANCE	Account balance in the deposit currency	Ac
ACCOUNT_COMPANY	Name of a company that serves the account	Ac
ACCOUNT_CREDIT	Account credit in the deposit currency	Ac
ACCOUNT_CURRENCY	Account currency	Ac
ACCOUNT_EQUITY	Account equity in the deposit currency	Ac
ACCOUNT_MARGIN_FREE	Free margin of an account in the deposit currency	Ac
ACCOUNT_LEVERAGE	Account leverage	Ac
ACCOUNT_LIMIT_ORDERS	Maximum allowed number of active pending orders (0-unlimited)	Ac

ACCOUNT_LOGIN	Account number	Ac
ACCOUNT_MARGIN	Account margin used in the deposit currency	Ac
ACCOUNT_MARGIN_LEVEL	Account margin level in percents	Ac
ACCOUNT_MARGIN_SO_CALL	Margin call level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	Ac
ACCOUNT_MARGIN_SO_MODE	Mode for setting the minimal allowed margin	Ac
ACCOUNT_MARGIN_SO_SO	Margin stop out level. Depending on the set ACCOUNT_MARGIN_SO_MODE is expressed in percents or in the deposit currency	Ac
ACCOUNT_NAME	Client name	Ac
ACCOUNT_PROFIT	Current profit of an account in the deposit currency	Ac
ACCOUNT_SERVER	Trade server name	Ac
ACCOUNT_STOPOUT_MODE_MONEY	Account stop out mode in money	Ac
ACCOUNT_STOPOUT_MODE_PERCENT	Account stop out mode in percents	Ac
ACCOUNT_TRADE_ALLOWED	Allowed trade for the current account	Ac
ACCOUNT_TRADE_EXPERT	Allowed trade for an Expert Advisor	Ac
ACCOUNT_TRADE_MODE	Account trade mode	Ac
ACCOUNT_TRADE_MODE_CONTEST	Contest account	Ac
ACCOUNT_TRADE_MODE_DEMO	Demo account	Ac
ACCOUNT_TRADE_MODE_REAL	Real account	Ac
ALIGN_CENTER	Centered (only for the Edit object)	Ob Ob Ch
ALIGN_LEFT	Left alignment	Ob Ob Ch
ALIGN_RIGHT	Right alignment	Ob Ob Ch
ANCHOR_BOTTOM	Anchor on the bottom side	Ob

		Ob
ANCHOR_CENTER	Anchor point strictly in the center of the object	Ob Ob
ANCHOR_LEFT	Anchor point to the left in the center	Ob Ob
ANCHOR_LEFT_LOWER	Anchor point at the lower left corner	Ob Ob
ANCHOR_LEFT_UPPER	Anchor point at the upper left corner	Ob Ob
ANCHOR_LOWER	Anchor point below in the center	Ob Ob
ANCHOR_RIGHT	Anchor point to the right in the center	Ob Ob
ANCHOR_RIGHT_LOWER	Anchor point at the lower right corner	Ob Ob
ANCHOR_RIGHT_UPPER	Anchor point at the upper right corner	Ob Ob
ANCHOR_TOP	Anchor on the top side	Ob Ob
ANCHOR_UPPER	Anchor point above in the center	Ob Ob
BORDER_FLAT	Flat form	Ob Ob
BORDER_RAISED	Prominent form	Ob Ob
BORDER_SUNKEN	Concave form	Ob Ob
CHAR_MAX	Maximal value, which can be represented by char type	Nu Cc
CHAR_MIN	Minimal value, which can be represented by char type	Nu Cc
CHART_AUTOSCROLL	Mode of automatic moving to the right border of the chart	Ch Ch
CHART_BARS	Display as a sequence of bars	Ch
CHART_BEGIN	Chart beginning (the oldest prices)	Ch
CHART_BRING_TO_TOP	Show chart on top of other charts	Ch

		Ch
CHART_CANDLES	Display as Japanese candlesticks	Ch
CHART_COLOR_ASK	Ask price level color	Ch Ch
CHART_COLOR_BACKGROUND	Chart background color	Ch Ch
CHART_COLOR_BID	Bid price level color	Ch Ch
CHART_COLOR_CANDLE_BEAR	Body color of a bear candlestick	Ch Ch
CHART_COLOR_CANDLE_BULL	Body color of a bull candlestick	Ch Ch
CHART_COLOR_CHART_DOWN	Color for the down bar, shadows and body borders of bear candlesticks	Ch Ch
CHART_COLOR_CHART_LINE	Line chart color and color of "Doji" Japanese candlesticks	Ch Ch
CHART_COLOR_CHART_UP	Color for the up bar, shadows and body borders of bull candlesticks	Ch Ch
CHART_COLOR_FOREGROUND	Color of axes, scales and OHLC line	Ch Ch
CHART_COLOR_GRID	Grid color	Ch Ch
CHART_COLOR_LAST	Line color of the last executed deal price (Last)	Ch Ch
CHART_COLOR_STOP_LEVEL	Color of stop order levels (Stop Loss and Take Profit)	Ch Ch
CHART_COLOR_VOLUME	Color of volumes and order opening levels	Ch Ch
CHART_COMMENT	Text of a comment in a chart	Ch Ch
CHART_CURRENT_POS	Current position	Ch
CHART_DRAG_TRADE_LEVELS	Permission to drag trading levels on a chart with a mouse. The drag mode is enabled by default (true value)	Ch Ch
CHART_END	Chart end (the latest prices)	Ch
CHART_EVENT_MOUSE_MOVE	Send notifications of mouse move and	Ch

	mouse click events (CHARTEVENT_MOUSE_MOVE) to all mql4 programs on a chart	Ch
CHART_EVENT_OBJECT_CREATE	Send a notification of an event of new object creation (CHARTEVENT_OBJECT_CREATE) to all mql4-programs on a chart	Ch Ch
CHART_EVENT_OBJECT_DELETE	Send a notification of an event of object deletion (CHARTEVENT_OBJECT_DELETE) to all mql4-programs on a chart	Ch Ch
CHART_FIRST_VISIBLE_BAR	Number of the first visible bar in the chart. Indexing of bars is the same as for timeseries.	Ch Ch
CHART_FIXED_MAX	Fixed chart maximum	Ch Ch
CHART_FIXED_MIN	Fixed chart minimum	Ch Ch
CHART_FIXED_POSITION	Chart fixed position from the left border in percent value. Chart fixed position is marked by a small gray triangle on the horizontal time axis. It is displayed only if the automatic chart scrolling to the right on tick incoming is disabled (see CHART_AUTOSCROLL property). The bar on a fixed position remains in the same place when zooming in and out.	Ch Ch
CHART_FOREGROUND	Price chart in the foreground	Ch Ch
CHART_HEIGHT_IN_PIXELS	Chart height in pixels	Ch Ch
CHART_IS_OFFLINE	Flag, indicating that chart opened in offline mode	Ch Ch
CHART_LINE	Display as a line drawn by Close prices	Ch
CHART_MODE	Chart type (candlesticks, bars or line)	Ch Ch
CHART_MOUSE_SCROLL	Scrolling the chart horizontally using the left mouse button. Vertical scrolling	Ch Ch

	is also available if the value of any following properties is set to true: CHART_SCALEFIX, CHART_SCALEFIX_11 or CHART_SCALE_PT_PER_BAR	
CHART_POINTS_PER_BAR	Scale in points per bar	Ch Ch
CHART_PRICE_MAX	Chart maximum	Ch Ch
CHART_PRICE_MIN	Chart minimum	Ch Ch
CHART_SCALE	Scale	Ch Ch
CHART_SCALE_PT_PER_BAR	Scale to be specified in points per bar	Ch Ch
CHART_SCALEFIX	Fixed scale mode	Ch Ch
CHART_SCALEFIX_11	Scale 1:1 mode	Ch Ch
CHART_SHIFT	Mode of price chart indent from the right border	Ch Ch
CHART_SHIFT_SIZE	The size of the zero bar indent from the right border in percents	Ch Ch
CHART_SHOW_ASK_LINE	Display Ask values as a horizontal line in a chart	Ch Ch
CHART_SHOW_BID_LINE	Display Bid values as a horizontal line in a chart	Ch Ch
CHART_SHOW_DATE_SCALE	Showing the time scale on a chart	Ch Ch
CHART_SHOW_GRID	Display grid in the chart	Ch Ch
CHART_SHOW_LAST_LINE	Display Last values as a horizontal line in a chart	Ch Ch
CHART_SHOW_OBJECT_DESCR	Pop-up descriptions of graphical objects	Ch Ch
CHART_SHOW_OHLC	Show OHLC values in the upper left corner	Ch Ch
CHART_SHOW_PERIOD_SEP	Display vertical separators between	Ch

	adjacent periods	Ch
CHART_SHOW_PRICE_SCALE	Showing the price scale on a chart	Ch Ch
CHART_SHOW_TRADE_LEVELS	Displaying trade levels in the chart (levels of open orders, Stop Loss, Take Profit and pending orders)	Ch Ch
CHART_SHOW_VOLUMES	Display volume in the chart	Ch Ch
CHART_VISIBLE_BARS	The number of bars on the chart that can be displayed	Ch Ch
CHART_VOLUME_HIDE	Volumes are not shown	Ch
CHART_VOLUME_TICK	Tick volumes	Ch
CHART_WIDTH_IN_BARS	Chart width in bars	Ch Ch
CHART_WIDTH_IN_PIXELS	Chart width in pixels	Ch Ch
CHART_WINDOW_HANDLE	Chart window handle (HWND)	Ch Ch
CHART_WINDOW_IS_VISIBLE	Visibility of subwindows	Ch Ch
CHART_WINDOW_YDISTANCE	The distance between the upper frame of the indicator subwindow and the upper frame of the main chart window, along the vertical Y axis, in pixels. In case of a mouse event, the cursor coordinates are passed in terms of the coordinates of the main chart window, while the coordinates of graphical objects in an indicator subwindow are set relative to the upper left corner of the subwindow. The value is required for converting the absolute coordinates of the main chart to the local coordinates of a subwindow for correct work with the graphical objects, whose coordinates are set relative to the upper left corner of the subwindow frame.	Ch Ch
CHART_WINDOWS_TOTAL	The total number of chart windows,	Ch

	including indicator subwindows	Ch
CHARTEVENT_CHART_CHANGE	Change of the chart size or modification of chart properties through the Properties dialog	On
CHARTEVENT_CLICK	Clicking on a chart	On
CHARTEVENT_CUSTOM	Initial number of an event from a range of custom events	On
CHARTEVENT_CUSTOM_LAST	The final number of an event from a range of custom events	On
CHARTEVENT_KEYDOWN	Keystrokes	On
CHARTEVENT_MOUSE_MOVE	Mouse move, mouse clicks (if CHART_EVENT_MOUSE_MOVE=true is set for the chart)	On
CHARTEVENT_OBJECT_CHANGE	Graphical object property changed via the properties dialog	On
CHARTEVENT_OBJECT_CLICK	Clicking on a graphical object	On
CHARTEVENT_OBJECT_CREATE	Graphical object created (if CHART_EVENT_OBJECT_CREATE=true is set for the chart)	On
CHARTEVENT_OBJECT_DELETE	Graphical object deleted (if CHART_EVENT_OBJECT_DELETE=true is set for the chart)	On
CHARTEVENT_OBJECT_DRAG	Drag and drop of a graphical object	On
CHARTEVENT_OBJECT_ENDEDIT	End of text editing in the graphical object Edit	On
CHARTS_MAX	The maximum possible number of simultaneously open charts in the terminal	Ot
CLR_NONE, clrNONE	Absence of color. Indicates empty state of colors	Ot
clrAliceBlue	Alice Blue	We
clrAntiqueWhite	Antique White	We
clrAqua	Aqua	We
clrAquamarine	Aquamarine	We
clrBeige	Beige	We
clrBisque	Bisque	We

clrBlack	Black	We
clrBlanchedAlmond	Blanched Almond	We
clrBlue	Blue	We
clrBlueViolet	Blue Violet	We
clrBrown	Brown	We
clrBurlyWood	Burly Wood	We
clrCadetBlue	Cadet Blue	We
clrChartreuse	Chartreuse	We
clrChocolate	Chocolate	We
clrCoral	Coral	We
clrCornflowerBlue	Cornflower Blue	We
clrCornsilk	Cornsilk	We
clrCrimson	Crimson	We
clrDarkBlue	Dark Blue	We
clrDarkGoldenrod	Dark Goldenrod	We
clrDarkGray	Dark Gray	We
clrDarkGreen	Dark Green	We
clrDarkKhaki	Dark Khaki	We
clrDarkOliveGreen	Dark Olive Green	We
clrDarkOrange	Dark Orange	We
clrDarkOrchid	Dark Orchid	We
clrDarkSalmon	Dark Salmon	We
clrDarkSeaGreen	Dark Sea Green	We
clrDarkSlateBlue	Dark Slate Blue	We
clrDarkSlateGray	Dark Slate Gray	We
clrDarkTurquoise	Dark Turquoise	We
clrDarkViolet	Dark Violet	We
clrDeepPink	Deep Pink	We
clrDeepSkyBlue	Deep Sky Blue	We
clrDimGray	Dim Gray	We

clrDodgerBlue	Dodger Blue	We
clrFireBrick	Fire Brick	We
clrForestGreen	Forest Green	We
clrGainsboro	Gainsboro	We
clrGold	Gold	We
clrGoldenrod	Goldenrod	We
clrGray	Gray	We
clrGreen	Green	We
clrGreenYellow	Green Yellow	We
clrHoneydew	Honeydew	We
clrHotPink	Hot Pink	We
clrIndianRed	Indian Red	We
clrIndigo	Indigo	We
clrIvory	Ivory	We
clrKhaki	Khaki	We
clrLavender	Lavender	We
clrLavenderBlush	Lavender Blush	We
clrLawnGreen	Lawn Green	We
clrLemonChiffon	Lemon Chiffon	We
clrLightBlue	Light Blue	We
clrLightCoral	Light Coral	We
clrLightCyan	Light Cyan	We
clrLightGoldenrod	Light Goldenrod	We
clrLightGray	Light Gray	We
clrLightGreen	Light Green	We
clrLightPink	Light Pink	We
clrLightSalmon	Light Salmon	We
clrLightSeaGreen	Light Sea Green	We
clrLightSkyBlue	Light Sky Blue	We
clrLightSlateGray	Light Slate Gray	We
clrLightSteelBlue	Light Steel Blue	We

clrLightYellow	Light Yellow	We
clrLime	Lime	We
clrLimeGreen	Lime Green	We
clrLinen	Linen	We
clrMagenta	Magenta	We
clrMaroon	Maroon	We
clrMediumAquamarine	Medium Aquamarine	We
clrMediumBlue	Medium Blue	We
clrMediumOrchid	Medium Orchid	We
clrMediumPurple	Medium Purple	We
clrMediumSeaGreen	Medium Sea Green	We
clrMediumSlateBlue	Medium Slate Blue	We
clrMediumSpringGreen	Medium Spring Green	We
clrMediumTurquoise	Medium Turquoise	We
clrMediumVioletRed	Medium Violet Red	We
clrMidnightBlue	Midnight Blue	We
clrMintCream	Mint Cream	We
clrMistyRose	Misty Rose	We
clrMoccasin	Moccasin	We
clrNavajoWhite	Navajo White	We
clrNavy	Navy	We
clrOldLace	Old Lace	We
clrOlive	Olive	We
clrOliveDrab	Olive Drab	We
clrOrange	Orange	We
clrOrangeRed	Orange Red	We
clrOrchid	Orchid	We
clrPaleGoldenrod	Pale Goldenrod	We
clrPaleGreen	Pale Green	We
clrPaleTurquoise	Pale Turquoise	We

clrPaleVioletRed	Pale Violet Red	We
clrPapayaWhip	Papaya Whip	We
clrPeachPuff	Peach Puff	We
clrPeru	Peru	We
clrPink	Pink	We
clrPlum	Plum	We
clrPowderBlue	Powder Blue	We
clrPurple	Purple	We
clrRed	Red	We
clrRosyBrown	Rosy Brown	We
clrRoyalBlue	Royal Blue	We
clrSaddleBrown	Saddle Brown	We
clrSalmon	Salmon	We
clrSandyBrown	Sandy Brown	We
clrSeaGreen	Sea Green	We
clrSeashell	Seashell	We
clrSienna	Sienna	We
clrSilver	Silver	We
clrSkyBlue	Sky Blue	We
clrSlateBlue	Slate Blue	We
clrSlateGray	Slate Gray	We
clrSnow	Snow	We
clrSpringGreen	Spring Green	We
clrSteelBlue	Steel Blue	We
clrTan	Tan	We
clrTeal	Teal	We
clrThistle	Thistle	We
clrTomato	Tomato	We
clrTurquoise	Turquoise	We
clrViolet	Violet	We

clrWheat	Wheat	We
clrWhite	White	We
clrWhiteSmoke	White Smoke	We
clrYellow	Yellow	We
clrYellowGreen	Yellow Green	We
CORNER_LEFT_LOWER	Center of coordinates is in the lower left corner of the chart	Ob Ob
CORNER_LEFT_UPPER	Center of coordinates is in the upper left corner of the chart	Ob Ob
CORNER_RIGHT_LOWER	Center of coordinates is in the lower right corner of the chart	Ob Ob
CORNER_RIGHT_UPPER	Center of coordinates is in the upper right corner of the chart	Ob Ob
CP_ACP	The current Windows ANSI code page.	Ch St Fi
CP_MACCP	The current system Macintosh code page. Note: This value is mostly used in earlier created program codes and is of no use now, since modern Macintosh computers use Unicode for encoding.	Ch St Fi
CP_OEMCP	The current system OEM code page.	Ch St Fi
CP_SYMBOL	Symbol code page	Ch St Fi
CP_THREAD_ACP	The Windows ANSI code page for the current thread.	Ch St Fi
CP_UTF7	UTF-7 code page.	Ch St Fi
CP_UTF8	UTF-8 code page.	Ch St Fi

CRYPT_AES128	AES encryption with 128 bit key (16 bytes)	Cr Cr
CRYPT_AES256	AES encryption with 256 bit key (32 bytes)	Cr Cr
CRYPT_ARCH_ZIP	ZIP archives	Cr Cr
CRYPT_BASE64	BASE64	Cr Cr
CRYPT_DES	DES encryption with 56 bit key (7 bytes)	Cr Cr
CRYPT_HASH_MD5	MD5 HASH calculation	Cr Cr
CRYPT_HASH_SHA1	SHA1 HASH calculation	Cr Cr
CRYPT_HASH_SHA256	SHA256 HASH calculation	Cr Cr
DBL_DIG	Number of significant decimal digits for double type	Nu Co
DBL_EPSILON	Minimal value, which satisfies the condition: 1.0+DBL_EPSILON != 1.0 (for double type)	Nu Co
DBL_MANT_DIG	Bits count in a mantissa for double type	Nu Co
DBL_MAX	Maximal value, which can be represented by double type	Nu Co
DBL_MAX_10_EXP	Maximal decimal value of exponent degree for double type	Nu Co
DBL_MAX_EXP	Maximal binary value of exponent degree for double type	Nu Co
DBL_MIN	Minimal positive value, which can be represented by double type	Nu Co
DBL_MIN_10_EXP	Minimal decimal value of exponent degree for double type	Nu Co
DBL_MIN_EXP	Minimal binary value of exponent degree for double type	Nu Co

DRAW_ARROW	Drawing arrows (symbols)	Dr
DRAW_HISTOGRAM	Drawing histogram	Dr
DRAW_LINE	Drawing line	Dr
DRAW_NONE	No drawing	Dr
DRAW_SECTION	Drawing sections	Dr
DRAW_ZIGZAG	Drawing sections between even and odd indicator buffers, 2 buffers of values	Dr
EMPTY	Indicates empty state of the parameter	Ot
EMPTY_VALUE	Empty value in an indicator buffer. Default custom indicator empty value	Ot
ERR_ACCOUNT_DISABLED	Account disabled	Ge
ERR_ARRAY_AS_PARAMETER_EXPECTED	Array as parameter expected	Ge
ERR_ARRAY_INDEX_OUT_OF_RANGE	Array index is out of range	Ge
ERR_ARRAY_INVALID	Invalid array	Ge
ERR_BROKER_BUSY	Broker is busy	Ge
ERR_CANNOT_CALL_FUNCTION	Cannot call function	Ge
ERR_CANNOT_LOAD_LIBRARY	Cannot load library	Ge
ERR_CANNOT_OPEN_FILE	Cannot open file	Ge
ERR_CHART_NOREPLY	No reply from chart	Ge
ERR_CHART_NOT_FOUND	Chart not found	Ge
ERR_CHART_PROP_INVALID	Unknown chart property	Ge
ERR_CHARTINDICATOR_NOT_FOUND	Chart indicator not found	Ge
ERR_CHARTWINDOW_NOT_FOUND	Chart subwindow not found	Ge
ERR_COMMON_ERROR	Common error	Ge
ERR_CUSTOM_INDICATOR_ERROR	Custom indicator error	Ge
ERR_DLL_CALLS_NOT_ALLOWED	DLL calls are not allowed	Ge
ERR_DLLFUNC_CRITICALERROR	DLL-function call critical error	Ge
ERR_DOUBLE_PARAMETER_EXPECTED	Double parameter expected	Ge
ERR_END_OF_FILE	End of file	Ge
ERR_EXTERNAL_CALLS_NOT_ALLOWED	Expert function calls are not allowed	Ge
ERR_FILE_ARRAYRESIZE_ERROR	Array resize error	Ge

ERR_FILE_BIN_STRINGSIZE	String size must be specified for binary file	Ge
ERR_FILE_BUFFER_ALLOCATION_ERROR	Text file buffer allocation error	Ge
ERR_FILE_CANNOT_CLEAN_DIRECTORY	Cannot clean directory	Ge
ERR_FILE_CANNOT_DELETE	Cannot delete file	Ge
ERR_FILE_CANNOT_DELETE_DIRECTORY	Cannot delete directory	Ge
ERR_FILE_CANNOT_OPEN	Cannot open file	Ge
ERR_FILE_CANNOT_REWRITE	File cannot be rewritten	Ge
ERR_FILE_DIRECTORY_NOT_EXIST	Directory does not exist	Ge
ERR_FILE_INCOMPATIBLE	Incompatible file (for string arrays-TXT, for others-BIN)	Ge
ERR_FILE_INVALID_HANDLE	Invalid file handle (file closed or was not opened)	Ge
ERR_FILE_IS_DIRECTORY	File is directory not file	Ge
ERR_FILE_NOT_BIN	File must be opened with FILE_BIN flag	Ge
ERR_FILE_NOT_CSV	File must be opened with FILE_CSV flag	Ge
ERR_FILE_NOT_DIRECTORY	Specified file is not directory	Ge
ERR_FILE_NOT_EXIST	File does not exist	Ge
ERR_FILE_NOT_TOREAD	File must be opened with FILE_READ flag	Ge
ERR_FILE_NOT_TOWRITE	File must be opened with FILE_WRITE flag	Ge
ERR_FILE_NOT_TXT	File must be opened with FILE_TXT flag	Ge
ERR_FILE_NOT_TXTORCSV	File must be opened with FILE_TXT or FILE_CSV flag	Ge
ERR_FILE_READ_ERROR	File read error	Ge
ERR_FILE_STRINGRESIZE_ERROR	String resize error	Ge
ERR_FILE_STRUCT_WITH_OBJECTS	Structure contains strings or dynamic arrays	Ge
ERR_FILE_TOO_LONG_FILENAME	Too long file name	Ge
ERR_FILE_TOO_MANY_OPENED	Too many opened files	Ge
ERR_FILE_WRITE_ERROR	File write error	Ge
ERR_FILE_WRONG_DIRECTORYNAME	Wrong directory name	Ge

ERR_FILE_WRONG_FILENAME	Wrong file name	Ge
ERR_FILE_WRONG_HANDLE	Wrong file handle (handle index is out of handle table)	Ge
ERR_FORMAT_TOO_MANY_FORMATTERS	Too many formatters in the format function	Ge
ERR_FORMAT_TOO_MANY_PARAMETERS	Parameters count exceeds formatters count	Ge
ERR_FUNC_NOT_ALLOWED_IN_TESTING	Function is not allowed in testing mode	Ge
ERR_FUNCTION_NOT_CONFIRMED	Function is not allowed for call	Ge
ERR_GLOBAL_VARIABLE_NOT_FOUND	Global variable not found	Ge
ERR_GLOBAL_VARIABLES_PROCESSING	Global variables processing error	Ge
ERR_HISTORY_WILL_UPDATED	Requested history data is in updating state	Ge
ERR_INCOMPATIBLE_ARRAYS	Arrays are incompatible	Ge
ERR_INCOMPATIBLE_FILEACCESS	Incompatible access to a file	Ge
ERR_INCORRECT_SERIESARRAY_USING	Incorrect series array using	Ge
ERR_INDICATOR_CANNOT_INIT	Custom indicator cannot initialize	Ge
ERR_INDICATOR_CANNOT_LOAD	Cannot load custom indicator	Ge
ERR_INTEGER_PARAMETER_EXPECTED	Integer parameter expected	Ge
ERR_INTERNAL_ERROR	Internal error	Ge
ERR_INVALID_ACCOUNT	Invalid account	Ge
ERR_INVALID_FUNCTION_PARAMSCNT	Invalid function parameters count	Ge
ERR_INVALID_FUNCTION_PARAMVALUE	Invalid function parameter value	Ge
ERR_INVALID_POINTER	Invalid pointer	Ge
ERR_INVALID_PRICE	Invalid price	Ge
ERR_INVALID_PRICE_PARAM	Invalid price	Ge
ERR_INVALID_STOPS	Invalid stops	Ge
ERR_INVALID_TICKET	Invalid ticket	Ge
ERR_INVALID_TRADE_PARAMETERS	Invalid trade parameters	Ge
ERR_INVALID_TRADE_VOLUME	Invalid trade volume	Ge
ERR_LONG_POSITIONS_ONLY_ALLOWED	Buy orders only allowed	Ge
ERR_LONGS_NOT_ALLOWED	Longs are not allowed. Check the Expert	Ge

	Advisor properties	
ERR_MALFUNCTIONAL_TRADE	Malfunctional trade operation	Ge
ERR_MARKET_CLOSED	Market is closed	Ge
ERR_NO_CONNECTION	No connection with trade server	Ge
ERR_NO_ERROR	No error returned	Ge
ERR_NO_HISTORY_DATA	No history data	Ge
ERR_NO_MEMORY_FOR_ARRAYSTRING	No memory for array string	Ge
ERR_NO_MEMORY_FOR_CALL_STACK	No memory for function call stack	Ge
ERR_NO_MEMORY_FOR_HISTORY	No memory for history data	Ge
ERR_NO_MEMORY_FOR_PARAM_STRING	No memory for parameter string	Ge
ERR_NO_MEMORY_FOR_RETURNED_STR	Not enough memory for temp string returned from function	Ge
ERR_NO_MEMORY_FOR_TEMP_STRING	No memory for temp string	Ge
ERR_NO_MQLERROR	No error returned	Ge
ERR_NO_OBJECT_NAME	No object name	Ge
ERR_NO_ORDER_SELECTED	No order selected	Ge
ERR_NO_RESULT	No error returned, but the result is unknown	Ge
ERR_NO_SPECIFIED_SUBWINDOW	No specified subwindow	Ge
ERR_NOT_ENOUGH_MONEY	Not enough money	Ge
ERR_NOT_ENOUGH_RIGHTS	Not enough rights	Ge
ERR_NOT_ENOUGH_STACK_FOR_PARAM	Not enough stack for parameter	Ge
ERR_NOT_INITIALIZED_ARRAY	Not initialized array	Ge
ERR_NOT_INITIALIZED_ARRAYSTRING	Not initialized string in array	Ge
ERR_NOT_INITIALIZED_STRING	Not initialized string	Ge
ERR_NOTIFICATION_ERROR	Notification error	Ge
ERR_NOTIFICATION_PARAMETER	Notification parameter error	Ge
ERR_NOTIFICATION_SETTINGS	Notifications disabled	Ge
ERR_NOTIFICATION_TOO_FREQUENT	Notification send too frequent	Ge
ERR_OBJECT_ALREADY_EXISTS	Object already exists	Ge
ERR_OBJECT_COORDINATES_ERROR	Object coordinates error	Ge
ERR_OBJECT_DOES_NOT_EXIST	Object does not exist	Ge

ERR_OFF_QUOTES	Off quotes	Ge
ERR_OLD_VERSION	Old version of the client terminal	Ge
ERR_ORDER_LOCKED	Order is locked	Ge
ERR_OUT_OF_MEMORY	Out of memory	Ge
ERR_PRICE_CHANGED	Price changed	Ge
ERR_RECURSIVE_STACK_OVERFLOW	Recursive stack overflow	Ge
ERR_REMAINDER_FROM_ZERO_DIVIDE	Remainder from zero divide	Ge
ERR_REQUOTE	Requote	Ge
ERR_RESOURCE_DUPLICATED	Duplicate resource	Ge
ERR_RESOURCE_NOT_FOUND	Resource not found	Ge
ERR_RESOURCE_NOT_SUPPORTED	Resource not supported	Ge
ERR_SEND_MAIL_ERROR	Send mail error	Ge
ERR_SERVER_BUSY	Trade server is busy	Ge
ERR_SHORTS_NOT_ALLOWED	Shorts are not allowed. Check the Expert Advisor properties	Ge
ERR_SOME_ARRAY_ERROR	Some array error	Ge
ERR_SOME_FILE_ERROR	Some file error	Ge
ERR_SOME_OBJECT_ERROR	Graphical object error	Ge
ERR_STRING_FUNCTION_INTERNAL	String function internal error	Ge
ERR_STRING_PARAMETER_EXPECTED	String parameter expected	Ge
ERR_SYMBOL_SELECT	Symbol select error	Ge
ERR_SYSTEM_BUSY	System is busy (never generated error)	Ge
ERR_TOO_FREQUENT_REQUESTS	Too frequent requests	Ge
ERR_TOO_LONG_STRING	Too long string	Ge
ERR_TOO_MANY_OPENED_FILES	Too many opened files	Ge
ERR_TOO_MANY_REQUESTS	Too many requests	Ge
ERR_TRADE_CONTEXT_BUSY	Trade context is busy	Ge
ERR_TRADE_DISABLED	Trade is disabled	Ge
ERR_TRADE_ERROR	Internal trade error	Ge
ERR_TRADE_EXPERT_DISABLED_BY_SERVER	Automated trading by Expert Advisors/Scripts disabled by trade server	Ge

ERR_TRADE_EXPIRATION_DENIED	Expirations are denied by broker	Ge
ERR_TRADE_HEDGE_PROHIBITED	An attempt to open an order opposite to the existing one when hedging is disabled	Ge
ERR_TRADE_MODIFY_DENIED	Modification denied because order is too close to market	Ge
ERR_TRADE_NOT_ALLOWED	Trade is not allowed. Enable checkbox "Allow live trading" in the Expert Advisor properties	Ge
ERR_TRADE_PROHIBITED_BY_FIFO	An attempt to close an order contravening the FIFO rule	Ge
ERR_TRADE_TIMEOUT	Trade timeout	Ge
ERR_TRADE_TOO_MANY_ORDERS	The amount of open and pending orders has reached the limit set by the broker	Ge
ERR_UNKNOWN_COMMAND	Unknown command	Ge
ERR_UNKNOWN_OBJECT_PROPERTY	Unknown object property	Ge
ERR_UNKNOWN_OBJECT_TYPE	Unknown object type	Ge
ERR_UNKNOWN_SYMBOL	Unknown symbol	Ge
ERR_USER_ERROR_FIRST	User defined errors start with this code	Ge
ERR_WEBREQUEST_CONNECT_FAILED	Failed to connect to specified URL	Ge
ERR_WEBREQUEST_INVALID_ADDRESS	Invalid URL	Ge
ERR_WEBREQUEST_REQUEST_FAILED	HTTP request failed	Ge
ERR_WEBREQUEST_TIMEOUT	Timeout exceeded	Ge
ERR_WRONG_FILE_NAME	Wrong file name	Ge
ERR_WRONG_FUNCTION_POINTER	Wrong function pointer	Ge
ERR_WRONG_JUMP	Wrong jump (never generated error)	Ge
ERR_ZERO_DIVIDE	Zero divide	Ge
FILE_ACCESS_DATE	Date of the last access to the file	Fi
FILE_ANSI	Strings of ANSI type (one byte symbols). Flag is used in FileOpen()	Fi
FILE_BIN	Binary read/write mode (without string to string conversion). Flag is used in FileOpen()	Fi

FILE_COMMON	The file path in the common folder of all client terminals \Terminal\Common\Files. Flag is used in FileOpen(), FileCopy(), FileMove() and in FileIsExist() functions.	Fi Fi
FILE_CREATE_DATE	Date of creation	Fi
FILE_CSV	CSV file (all its elements are converted to strings of the appropriate type, unicode or ansi, and separated by separator). Flag is used in FileOpen()	Fi
FILE_END	Get the end of file sign	Fi
FILE_EXISTS	Check the existence	Fi
FILE_IS_ANSI	The file is opened as ANSI (see FILE_ANSI)	Fi
FILE_IS_BINARY	The file is opened as a binary file (see FILE_BIN)	Fi
FILE_IS_COMMON	The file is opened in a shared folder of all terminals (see FILE_COMMON)	Fi
FILE_IS_CSV	The file is opened as CSV (see FILE_CSV)	Fi
FILE_IS_READABLE	The opened file is readable (see FILE_READ)	Fi
FILE_IS_TEXT	The file is opened as a text file (see FILE_TXT)	Fi
FILE_IS_WRITABLE	The opened file is writable (see FILE_WRITE)	Fi
FILE_LINE_END	Get the end of line sign	Fi
FILE_MODIFY_DATE	Date of the last modification	Fi
FILE_POSITION	Position of a pointer in the file	Fi
FILE_READ	File is opened for reading. Flag is used in FileOpen(). When opening a file specification of FILE_WRITE and/or FILE_READ is required.	Fi
FILE_REWRITE	Possibility for the file rewrite using functions FileCopy() and FileMove(). The file should exist or should be opened for writing, otherwise the file will not be opened.	Fi

FILE_SHARE_READ	Shared access for reading from several programs. Flag is used in FileOpen(), but it does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.	Fi
FILE_SHARE_WRITE	Shared access for writing from several programs. Flag is used in FileOpen(), but it does not replace the necessity to indicate FILE_WRITE and/or the FILE_READ flag when opening a file.	Fi
FILE_SIZE	File size in bytes	Fi
FILE_TXT	Simple text file (the same as csv file, but without taking into account the separators). Flag is used in FileOpen()	Fi
FILE_UNICODE	Strings of UNICODE type (two byte symbols). Flag is used in FileOpen()	Fi
FILE_WRITE	File is opened for writing. Flag is used in FileOpen(). When opening a file specification of FILE_WRITE and/or FILE_READ is required.	Fi
FLT_DIG	Number of significant decimal digits for float type	Nu Co
FLT_EPSILON	Minimal value, which satisfies the condition: 1.0+DBL_EPSILON != 1.0 (for float type)	Nu Co
FLT_MANT_DIG	Bits count in a mantissa for float type	Nu Co
FLT_MAX	Maximal value, which can be represented by float type	Nu Co
FLT_MAX_10_EXP	Maximal decimal value of exponent degree for float type	Nu Co
FLT_MAX_EXP	Maximal binary value of exponent degree for float type	Nu Co
FLT_MIN	Minimal positive value, which can be represented by float type	Nu Co
FLT_MIN_10_EXP	Minimal decimal value of exponent degree for float type	Nu Co
FLT_MIN_EXP	Minimal binary value of exponent	Nu

	degree for float type	Co
FRIDAY	Friday	Sy Sy
GANN_DOWN_TREND	Line corresponding to the downward trend	Ga
GANN_UP_TREND	Line corresponding to the uptrend line	Ga
IDABORT	"Abort" button has been pressed	Me
IDCANCEL	"Cancel" button has been pressed	Me
IDCONTINUE	"Continue" button has been pressed	Me
IDIGNORE	"Ignore" button has been pressed	Me
IDNO	"No" button has been pressed	Me
IDOK	"OK" button has been pressed	Me
IDRETRY	"Retry" button has been pressed	Me
IDTRYAGAIN	"Try Again" button has been pressed	Me
IDYES	"Yes" button has been pressed	Me
INDICATOR_DIGITS	Accuracy of drawing of indicator values	Cu Pr
INDICATOR_HEIGHT	Fixed height of the indicator's window (the preprocessor command #property indicator_height)	Cu Pr
INDICATOR_LEVELCOLOR	Color of the level line	Cu Pr
INDICATOR_LEVELS	Number of levels in the indicator window	Cu Pr
INDICATOR_LEVELSTYLE	Style of the level line	Cu Pr
INDICATOR_LEVELTEXT	Level description	Cu Pr
INDICATOR_LEVELVALUE	Level value	Cu Pr
INDICATOR_LEVELWIDTH	Thickness of the level line	Cu Pr
INDICATOR_MAXIMUM	Maximum of the indicator window	Cu Pr

INDICATOR_MINIMUM	Minimum of the indicator window	Cu Pr
INDICATOR_SHORTNAME	Short indicator name	Cu Pr
INT_MAX	Maximal value, which can be represented by int type	Nu Co
INT_MIN	Minimal value, which can be represented by int type	Nu Co
INVALID_HANDLE	Incorrect handle	Ot
IS_DEBUG_MODE	Flag that a mql4-program operates in debug mode	Ot
IS_PROFILE_MODE	Flag that a mql4-program operates in profiling mode	Ot
LONG_MAX	Maximal value, which can be represented by long type	Nu Co
LONG_MIN	Minimal value, which can be represented by long type	Nu Co
M_1_PI	1/pi	Ma
M_2_PI	2/pi	Ma
M_2_SQRTPI	2/sqrt(pi)	Ma
M_E	e	Ma
M_LN10	ln(10)	Ma
M_LN2	ln(2)	Ma
M_LOG10E	log10(e)	Ma
M_LOG2E	log2(e)	Ma
M_PI	pi	Ma
M_PI_2	pi/2	Ma
M_PI_4	pi/4	Ma
M_SQRT1_2	1/sqrt(2)	Ma
M_SQRT2	sqrt(2)	Ma
MB_ABORTRETRYIGNORE	Message window contains three buttons: Abort, Retry and Ignore	Me
MB_CANCELTRYCONTINUE	Message window contains three buttons: Cancel, Try Again, Continue	Me

MB_DEFBUTTON1	The first button MB_DEFBUTTON1 - is default, if the other buttons MB_DEFBUTTON2, MB_DEFBUTTON3, or MB_DEFBUTTON4 are not specified	Me
MB_DEFBUTTON2	The second button is default	Me
MB_DEFBUTTON3	The third button is default	Me
MB_DEFBUTTON4	The fourth button is default	Me
MB_ICONEXCLAMATION, MB_ICONWARNING	The exclamation/warning sign icon	Me
MB_ICONINFORMATION, MB_ICONASTERISK	The encircled i sign	Me
MB_ICONQUESTION	The question sign icon	Me
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	The STOP sign icon	Me
MB_OK	Message window contains only one button: OK. Default	Me
MB_OKCANCEL	Message window contains two buttons: OK and Cancel	Me
MB_RETRYCANCEL	Message window contains two buttons: Retry and Cancel	Me
MB_YESNO	Message window contains two buttons: Yes and No	Me
MB_YESNOCANCEL	Message window contains three buttons: Yes, No and Cancel	Me
MODE_ASK	Last incoming ask price. For the current symbol, it is stored in the predefined variable Ask	Ma
MODE_BID	Last incoming bid price. For the current symbol, it is stored in the predefined variable Bid	Ma
MODE_CHIKOUSPAN	Chikou Span	In
MODE_CLOSE	Close price	Se
MODE_DIGITS	Count of digits after decimal point in the symbol prices. For the current symbol, it is stored in the predefined variable Digits	Ma

MODE_EMA	Exponential averaging	Sn
MODE_EXPIRATION	Market expiration date (usually used for futures)	Ma
MODE_FREEZELEVEL	Order freeze level in points. If the execution price lies within the range defined by the freeze level, the order cannot be modified, cancelled or closed	Ma
MODE_GATORJAW	Jaw line	In
MODE_GATORLIPS	Lips line	In
MODE_GATORTEETH	Teeth line	In
MODE_HIGH	High day price	Se
MODE_HISTORY	An order is selected from closed and canceled orders	Or
MODE_KIJUNSEN	Kijun-sen	In
MODE_LOTSIZE	Lot size in the base currency	Ma
MODE_LOTSTEP	Step for changing lots	Ma
MODE_LOW	Low day price	Se
MODE_LOWER	Lower line	In
MODE_LWMA	Linear-weighted averaging	Sn
MODE_MAIN	Base indicator line	In
MODE_MARGINCALCMODE	Margin calculation mode. 0 - Forex; 1 - CFD; 2 - Futures; 3 - CFD for indices	Ma
MODE_MARGINHEDGED	Hedged margin calculated for 1 lot	Ma
MODE_MARGININIT	Initial margin requirements for 1 lot	Ma
MODE_MARGINMAINTENANCE	Margin to maintain open orders calculated for 1 lot	Ma
MODE_MARGINREQUIRED	Free margin required to open 1 lot for buying	Ma
MODE_MAXLOT	Maximum permitted amount of a lot	Ma
MODE_MINLOT	Minimum permitted amount of a lot	Ma
MODE_MINUSDI	-DI indicator line	In
MODE_OPEN	Open price	Se
MODE_PLUSDI	+DI indicator line	In

MODE_POINT	Point size in the quote currency. For the current symbol, it is stored in the predefined variable Point	Ma
MODE_PROFITCALCMODE	Profit calculation mode. 0 - Forex; 1 - CFD; 2 - Futures	Ma
MODE_SENKOUSPANA	Senkou Span A	In
MODE_SENKOUSPANB	Senkou Span B	In
MODE_SIGNAL	Signal line	In
MODE_SMA	Simple averaging	Sm
MODE_SMMA	Smoothed averaging	Sm
MODE_SPREAD	Spread value in points	Ma
MODE_STARTING	Market starting date (usually used for futures)	Ma
MODE_STOPLEVEL	Stop level in points A zero value of MODE_STOPLEVEL means either absence of any restrictions on the minimal distance for Stop Loss/Take Profit or the fact that a trade server utilizes some external mechanisms for dynamic level control, which cannot be translated in the client terminal. In the second case, GetLastError() can return error 130, because MODE_STOPLEVEL is actually "floating" here.	Ma
MODE_SWAPLONG	Swap of the buy order	Ma
MODE_SWAPSHORT	Swap of the sell order	Ma
MODE_SWAPTYPE	Swap calculation method. 0 - in points; 1 - in the symbol base currency; 2 - by interest; 3 - in the margin currency	Ma
MODE_TENKANSEN	Tenkan-sen	In
MODE_TICKSIZE	Tick size in points	Ma
MODE_TICKVALUE	Tick value in the deposit currency	Ma
MODE_TIME	The last incoming tick time (last known server time)	Se
MODE_TRADEALLOWED	Trade is allowed for the symbol	Ma

MODE_TRADES	An order is selected from open and pending orders	Or
MODE_UPPER	Upper line	In
MODE_VOLUME	Volume, used in iLowest() and iHighest() functions	Se
MONDAY	Monday	Sy Sy
MQL_CODEPAGE	Codepage used by an MQL4 program to output and convert strings (Print, PrintFormat, Alert, MessageBox, SendFTP, SendMail, SendNotification, etc.)	Ru Pr
MQL_DEBUG	The flag, that indicates the debug mode	M
MQL_DLLS_ALLOWED	The permission to use DLL for the given executed program	M
MQL_LICENSE_TYPE	Type of license of the EX4 module. The license refers to the EX4 module, from which a request is made using MQLInfoInteger(MQL_LICENSE_TYPE).	M
MQL_OPTIMIZATION	The flag, that indicates the optimization process	M
MQL_PROFILER	The flag, that indicates the program operating in the code profiling mode	M
MQL_PROGRAM_NAME	Name of the MQL4-program executed	M
MQL_PROGRAM_PATH	Path for the given executed program	M
MQL_PROGRAM_TYPE	Type of the MQL4 program	M
MQL_SIGNALS_ALLOWED	The permission to modify the Signals for the given executed program	M
MQL_TESTER	The flag, that indicates the tester process	M
MQL_TRADE_ALLOWED	The permission to trade for the given executed program	M
MQL_VISUAL_MODE	The flag, that indicates the visual tester process	M
NULL	Zero for any types. Also indicates empty state of the string	Ot

OBJ_ALL_PERIODS	The object is drawn in all timeframes	Vi
OBJ_ARROW	Arrow	Ob
OBJ_ARROW_BUY	Buy Sign	Ob
OBJ_ARROW_CHECK	Check Sign	Ob
OBJ_ARROW_DOWN	Arrow Down	Ob
OBJ_ARROW_LEFT_PRICE	Left Price Label	Ob
OBJ_ARROW_RIGHT_PRICE	Right Price Label	Ob
OBJ_ARROW_SELL	Sell Sign	Ob
OBJ_ARROW_STOP	Stop Sign	Ob
OBJ_ARROW_THUMB_DOWN	Thumbs Down	Ob
OBJ_ARROW_THUMB_UP	Thumbs Up	Ob
OBJ_ARROW_UP	Arrow Up	Ob
OBJ_BITMAP	Bitmap	Ob
OBJ_BITMAP_LABEL	Bitmap Label	Ob
OBJ_BUTTON	Button	Ob
OBJ_CHANNEL	Equidistant Channel	Ob
OBJ_CYCLES	Cycle Lines	Ob
OBJ_EDIT	Edit	Ob
OBJ_ELLIPSE	Ellipse	Ob
OBJ_EVENT	The "Event" object corresponding to an event in the economic calendar	Ob
OBJ_EXPANSION	Fibonacci Expansion	Ob
OBJ_FIBO	Fibonacci Retracement	Ob
OBJ_FIBOARC	Fibonacci Arcs	Ob
OBJ_FIBOCHANNEL	Fibonacci Channel	Ob
OBJ_FIBOFAN	Fibonacci Fan	Ob
OBJ_FIBOTIMES	Fibonacci Time Zones	Ob
OBJ_GANNFAN	Gann Fan	Ob
OBJ_GANNGRID	Gann Grid	Ob
OBJ_GANNLINE	Gann Line	Ob
OBJ_HLINE	Horizontal Line	Ob

OBJ_LABEL	Label	Ob
OBJ_NO_PERIODS, EMPTY	The object is not drawn in all timeframes	Vi
OBJ_PERIOD_D1	The object is drawn in day charts	Vi
OBJ_PERIOD_H1	The object is drawn in 1-hour chart	Vi
OBJ_PERIOD_H4	The object is drawn in 4-hour chart	Vi
OBJ_PERIOD_M1	The object is drawn in 1-minute chart	Vi
OBJ_PERIOD_M15	The object is drawn in 15-minute chart	Vi
OBJ_PERIOD_M30	The object is drawn in 30-minute chart	Vi
OBJ_PERIOD_M5	The object is drawn in 5-minute chart	Vi
OBJ_PERIOD_MN1	The object is drawn in month charts	Vi
OBJ_PERIOD_W1	The object is drawn in week charts	Vi
OBJ_PITCHFORK	Andrews Pitchfork	Ob
OBJ_RECTANGLE	Rectangle	Ob
OBJ_RECTANGLE_LABEL	The "Rectangle label" object for creating and designing the custom graphical interface.	Ob
OBJ_REGRESSION	Linear Regression Channel	Ob
OBJ_STDDEVCHANNEL	Standard Deviation Channel	Ob
OBJ_TEXT	Text	Ob
OBJ_TREND	Trend Line	Ob
OBJ_TRENDBYANGLE	Trend Line By Angle	Ob
OBJ_TRIANGLE	Triangle	Ob
OBJ_VLINE	Vertical Line	Ob
OBJPROP_ALIGN	Horizontal text alignment in the "Edit" object (OBJ_EDIT)	Ob Ob
OBJPROP_ANCHOR	Location of the anchor point of a graphical object	Ob Ob
OBJPROP_ANGLE	Double value to set/get angle object property in degrees	Ob
OBJPROP_ARROWCODE	Integer value or arrow enumeration to set/get arrow code object property	Ob
OBJPROP_BACK	Boolean value to set/get background	Ob

	drawing flag for object	
OBJPROP_BGCOLOR	The background color for OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL	Ob Ob
OBJPROP_BMPFILE	The name of BMP-file for Bitmap Label. See also Resources	Ob Ob
OBJPROP_BORDER_COLOR	Border color for the OBJ_EDIT and OBJ_BUTTON objects	Ob Ob
OBJPROP_BORDER_TYPE	Border type for the "Rectangle label" object	Ob Ob
OBJPROP_COLOR	Color value to set/get object color	Ob
OBJPROP_CORNER	Integer value to set/get anchor corner property for label objects. Must be from 0-3.	Ob
OBJPROP_CREATETIME	Time of object creation	Ob Ob
OBJPROP_DEVIATION	Double value to set/get deviation property for Standard deviation objects	Ob
OBJPROP_DRAWLINES	Displaying lines for marking the Elliott Wave	Ob Ob
OBJPROP_ELLIPSE	Boolean value to set/get ellipse flag for fibo arcs	Ob
OBJPROP_FIBOLEVELS	Integer value to set/get Fibonacci object level count. Can be from 0 to 32	Ob
OBJPROP_FIRSTLEVEL+n	Integer value to set/get the value of Fibonacci object level with index n. Index n can be from 0 (number of levels -1), but not larger than 31	Ob
OBJPROP_FONT	Font	Ob Ob
OBJPROP_FONTSIZE	Integer value to set/get font size for text objects	Ob
OBJPROP_HIDDEN	Prohibit showing of the name of a graphical object in the list of objects from the terminal menu "Charts" - "Objects" - "List of objects". The true value allows to hide an object from the list. By default, true is set to the objects that display calendar events,	Ob Ob

	trading history and to the objects created from MQL4 programs. To see such graphical objects and access their properties, click on the "All" button in the "List of objects" window.	
OBJPROP_LEVELCOLOR	Color value to set/get object level line color	Ob
OBJPROP_LEVELS	Number of levels	Ob Ob
OBJPROP_LEVELSTYLE	Value is one of STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT constants to set/get object level line style	Ob
OBJPROP_LEVELTEXT	Level description	Ob Ob
OBJPROP_LEVELVALUE	Level value	Ob Ob
OBJPROP_LEVELWIDTH	Integer value to set/get object level line width. Can be from 1 to 5	Ob
OBJPROP_NAME	Object name	Ob Ob
OBJPROP_PRICE	Price coordinate	Ob Ob
OBJPROP_PRICE1	Double value to set/get first coordinate price part	Ob
OBJPROP_PRICE2	Double value to set/get second coordinate price part	Ob
OBJPROP_PRICE3	Double value to set/get third coordinate price part	Ob
OBJPROP_RAY	Boolean value to set/get ray flag of object.	Ob
OBJPROP_RAY_RIGHT	Ray goes to the right	Ob Ob
OBJPROP_READONLY	Ability to edit text in the Edit object	Ob Ob
OBJPROP_SCALE	Double value to set/get scale object	Ob

	property	
OBJPROP_SELECTABLE	Object availability	Ob Ob
OBJPROP_SELECTED	Object is selected	Ob Ob
OBJPROP_STATE	Button state (pressed / depressed)	Ob Ob
OBJPROP_STYLE	Value is one of STYLE_SOLID, STYLE_DASH, STYLE_DOT, STYLE_DASHDOT, STYLE_DASHDOTDOT constants to set/get object line style	Ob
OBJPROP_SYMBOL	Symbol for the Chart object	Ob Ob
OBJPROP_TEXT	Description of the object (the text contained in the object)	Ob Ob
OBJPROP_TIME	Time coordinate	Ob Ob
OBJPROP_TIME1	Datetime value to set/get first coordinate time part	Ob
OBJPROP_TIME2	Datetime value to set/get second coordinate time part	Ob
OBJPROP_TIME3	Datetime value to set/get third coordinate time part	Ob
OBJPROP_TIMEFRAMES	Value can be one or combination (bitwise addition) of object visibility constants to set/get timeframe object property	Ob
OBJPROP_TOOLTIP	The text of a tooltip. If the property is not set, then the tooltip generated automatically by the terminal is shown. A tooltip can be disabled by setting the "\n" (line feed) value to it	Ob Ob
OBJPROP_TYPE	Object type	Ob Ob
OBJPROP_WIDTH	Integer value to set/get object line width. Can be from 1 to 5	Ob
OBJPROP_XDISTANCE	Integer value to set/get anchor X	Ob

	distance object property in pixels (see note)	
OBJPROP_XOFFSET	The X coordinate of the upper left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	Ob Ob
OBJPROP_XSIZE	The object's width along the X axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	Ob Ob
OBJPROP_YDISTANCE	Integer value is to set/get anchor Y distance object property in pixels (see note)	Ob
OBJPROP_YOFFSET	The Y coordinate of the upper left corner of the rectangular visible area in the graphical objects "Bitmap Label" and "Bitmap" (OBJ_BITMAP_LABEL and OBJ_BITMAP). The value is set in pixels relative to the upper left corner of the original image.	Ob Ob
OBJPROP_YSIZE	The object's height along the Y axis in pixels. Specified for OBJ_LABEL (read only), OBJ_BUTTON, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL objects.	Ob Ob
OBJPROP_ZORDER	Priority of a graphical object for receiving events of clicking on a chart (CHARTEVENT_CLICK). The default zero value is set when creating an object; the priority can be increased if necessary. When applying objects one over another, only one of them with the highest priority will receive the CHARTEVENT_CLICK event.	Ob Ob
OP_BUY	Buy operation	Or
OP_BUYLIMIT	Buy limit pending order	Or

OP_BUYSTOP	Buy stop pending order	Or
OP_SELL	Sell operation	Or
OP_SELLLIMIT	Sell limit pending order	Or
OP_SELLSTOP	Sell stop pending order	Or
PERIOD_CURRENT	Current timeframe	Ch
PERIOD_D1	1 day	Ch
PERIOD_H1	1 hour	Ch
PERIOD_H12	12 hours	Ch
PERIOD_H2	2 hours	Ch
PERIOD_H3	3 hours	Ch
PERIOD_H4	4 hours	Ch
PERIOD_H6	6 hours	Ch
PERIOD_H8	8 hours	Ch
PERIOD_M1	1 minute	Ch
PERIOD_M10	10 minutes	Ch
PERIOD_M12	12 minutes	Ch
PERIOD_M15	15 minutes	Ch
PERIOD_M2	2 minutes	Ch
PERIOD_M20	20 minutes	Ch
PERIOD_M3	3 minutes	Ch
PERIOD_M30	30 minutes	Ch
PERIOD_M4	4 minutes	Ch
PERIOD_M5	5 minutes	Ch
PERIOD_M6	6 minutes	Ch
PERIOD_MN1	1 month	Ch
PERIOD_W1	1 week	Ch
POINTER_AUTOMATIC	Pointer of any objects created automatically (not using new())	Ch
POINTER_DYNAMIC	Pointer of the object created by the new() operator	Ch
POINTER_INVALID	Incorrect pointer	Ch

PRICE_CLOSE	Close price	Pr
PRICE_HIGH	The maximum price for the period	Pr
PRICE_LOW	The minimum price for the period	Pr
PRICE_MEDIAN	Median price, (high + low)/2	Pr
PRICE_OPEN	Open price	Pr
PRICE_TYPICAL	Typical price, (high + low + close)/3	Pr
PRICE_WEIGHTED	Average price, (high + low + close + close)/4	Pr
REASON_ACCOUNT	Another account has been activated or reconnection to the trade server has occurred due to changes in the account settings	Ur
REASON_CHARTCHANGE	Symbol or chart period has been changed	Ur
REASON_CHARTCLOSE	Chart has been closed	Ur
REASON_CLOSE	Terminal has been closed	Ur
REASON_INITFAILED	This value means that OnInit() handler has returned a nonzero value	Ur
REASON_PARAMETERS	Input parameters have been changed by a user	Ur
REASON_PROGRAM	Expert Advisor terminated its operation by calling the ExpertRemove() function	Ur
REASON_RECOMPILE	Program has been recompiled	Ur
REASON_REMOVE	Program has been deleted from the chart	Ur
REASON_TEMPLATE	A new template has been applied	Ur
SATURDAY	Saturday	Sy Sy
SEEK_CUR	Current position of a file pointer	Fi
SEEK_END	File end	Fi
SEEK_SET	File beginning	Fi
SELECT_BY_POS	An order is selected based on its position in the list of orders	Or
SELECT_BY_TICKET	An order is selected by its ticket	Or

SERIES_BARS_COUNT	Bars count for the symbol-period for the current moment	Se
SERIES_FIRSTDATE	The very first date for the symbol-period for the current moment	Se
SERIES_LASTBAR_DATE	Open time of the last bar of the symbol-period	Se
SERIES_SERVER_FIRSTDATE	The very first date in the history of the symbol on the server regardless of the timeframe	Se
SHORT_MAX	Maximal value, which can be represented by short type	Nu Co
SHORT_MIN	Minimal value, which can be represented by short type	Nu Co
SIGNAL_BASE_AUTHOR_LOGIN	Author login	Si
SIGNAL_BASE_BALANCE	Account balance	Si
SIGNAL_BASE_BROKER	Broker name (company)	Si
SIGNAL_BASE_BROKER_SERVER	Broker server	Si
SIGNAL_BASE_CURRENCY	Signal base currency	Si
SIGNAL_BASE_DATE_PUBLISHED	Publication date (date when it become available for subscription)	Si
SIGNAL_BASE_DATE_STARTED	Monitoring starting date	Si
SIGNAL_BASE_EQUITY	Account equity	Si
SIGNAL_BASE_GAIN	Account gain	Si
SIGNAL_BASE_ID	Signal ID	Si
SIGNAL_BASE_LEVERAGE	Account leverage	Si
SIGNAL_BASE_MAX_DRAWDOWN	Account maximum drawdown	Si
SIGNAL_BASE_NAME	Signal name	Si
SIGNAL_BASE_PIPS	Profit in pips	Si
SIGNAL_BASE_PRICE	Signal subscription price	Si
SIGNAL_BASE_RATING	Position in rating	Si
SIGNAL_BASE_ROI	Return on Investment (%)	Si
SIGNAL_BASE_SUBSCRIBERS	Number of subscribers	Si
SIGNAL_BASE_TRADE_MODE	Account type (0-real, 1-demo, 2-contest)	Si

SIGNAL_BASE_TRADES	Number of trades	Si
SIGNAL_INFO_CONFIRMATIONS_DISABLED	The flag enables synchronization without confirmation dialog	Si Si
SIGNAL_INFO_COPY_SLTP	Copy Stop Loss and Take Profit flag	Si Si
SIGNAL_INFO_DEPOSIT_PERCENT	Deposit percent (%)	Si Si
SIGNAL_INFO_EQUITY_LIMIT	Equity limit	Si Si
SIGNAL_INFO_ID	Signal id, r/o	Si Si
SIGNAL_INFO_NAME	Signal name, r/o	Si
SIGNAL_INFO_SLIPPAGE	Slippage (used when placing market orders in synchronization of positions and copying of trades)	Si Si
SIGNAL_INFO_SUBSCRIPTION_ENABLED	"Copy trades by subscription" permission flag	Si Si
SIGNAL_INFO_TERMS_AGREE	"Agree to terms of use of Signals service" flag, r/o	Si Si
SIGNAL_INFO_VOLUME_PERCENT	Maximum percent of deposit used (%), r/o	Si Si
STAT_BALANCE_DD	Maximum balance drawdown in monetary terms. In the process of trading, a balance may have numerous drawdowns; here the largest value is taken	Te
STAT_BALANCE_DD_RELATIVE	Balance drawdown in monetary terms that was recorded at the moment of the maximum balance drawdown as a percentage (STAT_BALANCE_DDREL_PERCENT).	Te
STAT_BALANCE_DDREL_PERCENT	Maximum balance drawdown as a percentage. In the process of trading, a balance may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	Te

STAT_BALANCEDD_PERCENT	Balance drawdown as a percentage that was recorded at the moment of the maximum balance drawdown in monetary terms (STAT_BALANCE_DD).	Te
STAT_BALANCEMIN	Minimum balance value	Te
STAT_CONLOSSMAX	Maximum loss in a series of losing trades. The value is less than or equal to zero	Te
STAT_CONLOSSMAX_TRADES	The number of trades that have formed STAT_CONLOSSMAX (maximum loss in a series of losing trades)	Te
STAT_CONPROFITMAX	Maximum profit in a series of profitable trades. The value is greater than or equal to zero	Te
STAT_CONPROFITMAX_TRADES	The number of trades that have formed STAT_CONPROFITMAX (maximum profit in a series of profitable trades)	Te
STAT_CUSTOM_ONTESTER	The value of the calculated custom optimization criterion returned by the OnTester() function	Te
STAT_DEALS	The number of deals	Te
STAT_EQUITY_DD	Maximum equity drawdown in monetary terms. In the process of trading, numerous drawdowns may appear on the equity; here the largest value is taken	Te
STAT_EQUITY_DD_RELATIVE	Equity drawdown in monetary terms that was recorded at the moment of the maximum equity drawdown in percent (STAT_EQUITY_DDREL_PERCENT).	Te
STAT_EQUITY_DDREL_PERCENT	Maximum equity drawdown as a percentage. In the process of trading, an equity may have numerous drawdowns, for each of which the relative drawdown value in percents is calculated. The greatest value is returned	Te
STAT_EQUITYDD_PERCENT	Drawdown in percent that was recorded at the moment of the maximum equity	Te

	drawdown in monetary terms (STAT_EQUITY_DD).	
STAT_EQUITYMIN	Minimum equity value	Te
STAT_EXPECTED_PAYOFF	Expected payoff	Te
STAT_GROSS_LOSS	Total loss, the sum of all negative trades. The value is less than or equal to zero	Te
STAT_GROSS_PROFIT	Total profit, the sum of all profitable (positive) trades. The value is greater than or equal to zero	Te
STAT_INITIAL_DEPOSIT	The value of the initial deposit	Te
STAT_LONG_TRADES	Long trades	Te
STAT_LOSS_TRADES	Losing trades	Te
STAT_LOSSTRADES_AVGCON	Average length of a losing series of trades	Te
STAT_MAX_CONLOSS_TRADES	The number of trades in the longest series of losing trades STAT_MAX_CONLOSSES	Te
STAT_MAX_CONLOSSES	The total loss of the longest series of losing trades	Te
STAT_MAX_CONPROFIT_TRADES	The number of trades in the longest series of profitable trades STAT_MAX_CONWINS	Te
STAT_MAX_CONWINS	The total profit of the longest series of profitable trades	Te
STAT_MAX_LOSSTRADE	Maximum loss the lowest value of all losing trades. The value is less than or equal to zero	Te
STAT_MAX_PROFITTRADE	Maximum profit the largest value of all profitable trades. The value is greater than or equal to zero	Te
STAT_MIN_MARGINLEVEL	Minimum value of the margin level	Te
STAT_PROFIT	Net profit after testing, the sum of STAT_GROSS_PROFIT and STAT_GROSS_LOSS (STAT_GROSS_LOSS is always less than or equal to zero)	Te
STAT_PROFIT_FACTOR	Profit factor, equal to the ratio of	Te

	STAT_GROSS_PROFIT/STAT_GROSS_LOSS. If STAT_GROSS_LOSS=0, the profit factor is equal to DBL_MAX	
STAT_PROFIT_LONGTRADES	Profitable long trades	Te
STAT_PROFIT_SHORTTRADES	Profitable short trades	Te
STAT_PROFIT_TRADES	Profitable trades	Te
STAT_PROFITTRADES_AVGCON	Average length of a profitable series of trades	Te
STAT_RECOVERY_FACTOR	Recovery factor, equal to the ratio of STAT_PROFIT/STAT_BALANCE_DD	Te
STAT_SHARPE_RATIO	Sharpe ratio	Te
STAT_SHORT_TRADES	Short trades	Te
STAT_TRADES	The number of trades	Te
STAT_WITHDRAWAL	Money withdrawn from an account	Te
STO_CLOSECLOSE	Calculation is based on Close/Close prices	iSt
STO_LOWHIGH	Calculation is based on Low/High prices	iSt
STYLE_DASH	The pen is dashed	Dr
STYLE_DASHDOT	The pen has alternating dashes and dots	Dr
STYLE_DASHDOTDOT	The pen has alternating dashes and double dots	Dr
STYLE_DOT	The pen is dotted	Dr
STYLE_SOLID	The pen is solid	Dr
SUNDAY	Sunday	Sy Sy
SYMBOL_ARROWDOWN	Arrow down symbol	Ar
SYMBOL_ARROWUP	Arrow up symbol	Ar
SYMBOL_ASK	Ask - best buy offer	Sy
SYMBOL_ASKHIGH	Not supported	Sy
SYMBOL_ASKLOW	Not supported	Sy
SYMBOL_BID	Bid - best sell offer	Sy
SYMBOL_BIDHIGH	Not supported	Sy
SYMBOL_BIDLOW	Not supported	Sy

SYMBOL_CHECKSIGN	Check sign symbol	Ar
SYMBOL_CURRENCY_BASE	Basic currency of a symbol	Sy
SYMBOL_CURRENCY_MARGIN	Margin currency	Sy
SYMBOL_CURRENCY_PROFIT	Profit currency	Sy
SYMBOL_DESCRIPTION	Symbol description	Sy
SYMBOL_DIGITS	Digits after a decimal point	Sy
SYMBOL_EXPIRATION_MODE	Not supported	Sy
SYMBOL_EXPIRATION_TIME	Date of the symbol trade end (usually used for futures)	Sy
SYMBOL_FILLING_MODE	Not supported	Sy
SYMBOL_LAST	Not supported	Sy
SYMBOL_LASTHIGH	Not supported	Sy
SYMBOL_LASTLOW	Not supported	Sy
SYMBOL_LEFTPRICE	Left sided price label	Ar
SYMBOL_MARGIN_INITIAL	Initial margin means the amount in the margin currency required for opening an order with the volume of one lot. It is used for checking a client's assets when he or she enters the market.	Sy
SYMBOL_MARGIN_LIMIT	Not supported	Sy
SYMBOL_MARGIN_LONG	Not supported	Sy
SYMBOL_MARGIN_MAINTENANCE	The maintenance margin. If it is set, it sets the margin amount in the margin currency of the symbol, charged from one lot. It is used for checking a client's assets when his/her account state changes. If the maintenance margin is equal to 0, the initial margin is used.	Sy
SYMBOL_MARGIN_SHORT	Not supported	Sy
SYMBOL_MARGIN_STOP	Not supported	Sy
SYMBOL_MARGIN_STOPLIMIT	Not supported	Sy
SYMBOL_ORDER_MODE	Not supported	Sy
SYMBOL_PATH	Path in the symbol tree	Sy
SYMBOL_POINT	Symbol point value	Sy

SYMBOL_RIGHTPRICE	Right sided price label	Ar
SYMBOL_SELECT	Symbol is selected in Market Watch	Sy
SYMBOL_SESSION_AW	Not supported	Sy
SYMBOL_SESSION_BUY_ORDERS	Not supported	Sy
SYMBOL_SESSION_BUY_ORDERS_VOLUME	Not supported	Sy
SYMBOL_SESSION_CLOSE	Not supported	Sy
SYMBOL_SESSION_DEALS	Not supported	Sy
SYMBOL_SESSION_INTEREST	Not supported	Sy
SYMBOL_SESSION_OPEN	Not supported	Sy
SYMBOL_SESSION_PRICE_LIMIT_MAX	Not supported	Sy
SYMBOL_SESSION_PRICE_LIMIT_MIN	Not supported	Sy
SYMBOL_SESSION_PRICE_SETTLEMENT	Not supported	Sy
SYMBOL_SESSION_SELL_ORDERS	Not supported	Sy
SYMBOL_SESSION_SELL_ORDERS_VOLUME	Not supported	Sy
SYMBOL_SESSION_TURNOVER	Not supported	Sy
SYMBOL_SESSION_VOLUME	Not supported	Sy
SYMBOL_SPREAD	Spread value in points	Sy
SYMBOL_SPREAD_FLOAT	Indication of a floating spread	Sy
SYMBOL_START_TIME	Date of the symbol trade beginning (usually used for futures)	Sy
SYMBOL_STOPSIGN	Stop sign symbol	Ar
SYMBOL_SWAP_LONG	Buy order swap value	Sy
SYMBOL_SWAP_MODE	Swap calculation model	Sy
SYMBOL_SWAP_ROLLOVER3DAYS	Day of week to charge 3 days swap rollover	Sy
SYMBOL_SWAP_SHORT	Sell order swap value	Sy
SYMBOL_THUMBSDOWN	Thumb down symbol	Ar
SYMBOL_THUMBSUP	Thumb up symbol	Ar
SYMBOL_TIME	Time of the last quote	Sy
SYMBOL_TRADE_CALC_MODE	Contract price calculation mode	Sy

SYMBOL_TRADE_CONTRACT_SIZE	Trade contract size	Sy
SYMBOL_TRADE_EXECUTION_EXCHANGE	Exchange execution	Sy
SYMBOL_TRADE_EXECUTION_INSTANT	Instant execution	Sy
SYMBOL_TRADE_EXECUTION_MARKET	Market execution	Sy
SYMBOL_TRADE_EXECUTION_REQUEST	Execution by request	Sy
SYMBOL_TRADE_EXEMODE	Deal execution mode	Sy
SYMBOL_TRADE_FREEZE_LEVEL	Distance to freeze trade operations in points	Sy
SYMBOL_TRADE_MODE	Order execution type	Sy
SYMBOL_TRADE_MODE_CLOSEONLY	Allowed only position close operations	Sy
SYMBOL_TRADE_MODE_DISABLED	Trade is disabled for the symbol	Sy
SYMBOL_TRADE_MODE_FULL	No trade restrictions	Sy
SYMBOL_TRADE_MODE_LONGONLY	Allowed only long positions	Sy
SYMBOL_TRADE_MODE_SHORTONLY	Allowed only short positions	Sy
SYMBOL_TRADE_STOPS_LEVEL	Minimal indention in points from the current close price to place Stop orders	Sy
SYMBOL_TRADE_TICK_SIZE	Minimal price change	Sy
SYMBOL_TRADE_TICK_VALUE	Value of SYMBOL_TRADE_TICK_VALUE_PROFIT	Sy
SYMBOL_TRADE_TICK_VALUE_LOSS	Not supported	Sy
SYMBOL_TRADE_TICK_VALUE_PROFIT	Not supported	Sy
SYMBOL_VOLUME	Not supported	Sy
SYMBOL_VOLUME_LIMIT	Not supported	Sy
SYMBOL_VOLUME_MAX	Maximal volume for a deal	Sy
SYMBOL_VOLUME_MIN	Minimal volume for a deal	Sy
SYMBOL_VOLUME_STEP	Minimal volume change step for deal execution	Sy
SYMBOL_VOLUMEHIGH	Not supported	Sy
SYMBOL_VOLUMELOW	Not supported	Sy
TERMINAL_BUILD	The client terminal build number	Te
TERMINAL_CODEPAGE	Number of the code page of the language installed in the client terminal	Te

TERMINAL_COMMONDATA_PATH	Common path for all of the terminals installed on a computer	Te
TERMINAL_COMMUNITY_ACCOUNT	The flag indicates the presence of MQL5.community authorization data in the terminal	Te
TERMINAL_COMMUNITY_BALANCE	Balance in MQL5.community	Te
TERMINAL_COMMUNITY_CONNECTION	Connection to MQL5.community	Te
TERMINAL_COMPANY	Company name	Te
TERMINAL_CONNECTED	Connection to a trade server	Te
TERMINAL_CPU_CORES	The number of CPU cores in the system	Te
TERMINAL_DATA_PATH	Folder in which terminal data are stored	Te
TERMINAL_DISK_SPACE	Free disk space for the MQL4\Files folder of the terminal (agent), MB	Te
TERMINAL_DLLS_ALLOWED	Permission to use DLL	Te
TERMINAL_EMAIL_ENABLED	Permission to send e-mails using SMTP-server and login, specified in the terminal settings	Te
TERMINAL_FTP_ENABLED	Permission to send reports using FTP-server and login, specified in the terminal settings	Te
TERMINAL_LANGUAGE	Language of the terminal	Te
TERMINAL_MAXBARS	The maximal bars count on the chart	Te
TERMINAL_MEMORY_AVAILABLE	Free memory of the terminal (agent) process, MB	Te
TERMINAL_MEMORY_PHYSICAL	Physical memory in the system, MB	Te
TERMINAL_MEMORY_TOTAL	Memory available to the process of the terminal (agent), MB	Te
TERMINAL_MEMORY_USED	Memory used by the terminal (agent), MB	Te
TERMINAL_MQID	The flag indicates the presence of MetaQuotes ID data to send Push notifications	Te
TERMINAL_NAME	Terminal name	Te
TERMINAL_NOTIFICATIONS_ENABLED	Permission to send notifications to	Te

	smartphone	
TERMINAL_PATH	Folder from which the terminal is started	Te
TERMINAL_PING_LAST	The last known value of a ping to a trade server in microseconds. One second comprises of one million microseconds	Te
TERMINAL_SCREEN_DPI	The resolution of information display on the screen is measured as number of Dots in a line per Inch (DPI).	Te
TERMINAL_TRADE_ALLOWED	Permission to trade	Te
THURSDAY	Thursday	Sy Sy
TUESDAY	Tuesday	Sy Sy
UCHAR_MAX	Maximal value, which can be represented by uchar type	Nu Co
UINT_MAX	Maximal value, which can be represented by uint type	Nu Co
ULONG_MAX	Maximal value, which can be represented by ulong type	Nu Co
USHORT_MAX	Maximal value, which can be represented by ushort type	Nu Co
VOLUME_TICK	Tick volume	Pr
WEDNESDAY	Wednesday	Sy Sy
WHOLE_ARRAY	Used with array functions. Indicates that all array elements will be processed. Means the number of items remaining until the end of the array, i.e., the entire array will be processed	Ot
WRONG_VALUE	The constant can be implicitly cast to any enumeration type	Ot