Manual de referencia del lenguaje MQL 5

para el terminal de cliente MetaTrader 5

APRENDA el lenguaje MQL5 y REALICE cualquiera de estas tares:

- Creación de sus propios indicadores del análisis técnico de cualquier complejidad
- Autotrading – automatización del sistema comercial para operar en diferentes mercados financieros
- Diseño de herramientas analíticas a base de los avances matemáticos y métodos clásicos
- Desarrollo de sistemas informativo-comerciales para un amplio abanico de tareas (trading, monitoring, señales, etc.)

Contenido

Manual de referencia de MQL5

MetaQuotes Language 5 (MQL5) es un lenguaje built-in de programación de estrategias comerciales que ha sido desarrollado por la compañía MetaQuotes Software Corp., basándose en su amplia experiencia de varios años en el ámbito de creación de plataformas informativo-comerciales. Dicho lenguaje permite escribir sus propios programas-expertos (Expert Advisors) que hacen la gestión de procesos comerciales totalmente automatizada, y que son completamente adecuados para la realización de sus propias estrategias comerciales. Ademas, utilizando MQL5 se puede crear los indicadores técnicos personales (Custom Indicators), scripts (Scripts) y bibliotecas de funciones (Libraries).

MQL5 contiene la gran cantidad de funciones necesarias para el análisis de las cotizaciones actuales y recibidas anteriormente, en el programa están integrados los indicadores y funciones principales para la gestión de las posiciones comerciales y control de éstas.

Para escribir el código del programa se utiliza el procesador especializado de textos MetaEditor 5 que marca diferentes construcciones del lenguaje MQL5 con distintos colores, lo que permite al usuario orientarse mejor en el texto del sistema especializado. Como el sistema de consulta acerca del lenguaje MQL5 se utiliza el diccionario MetaQuotes Language Dictionary.

El manual de referencia contiene las funciones, operaciones, palabras reservadas y otras construcciones del lenguaje divididas por categorías. El manual permite averiguar la descripción de cada elemento que se utiliza y que compone el lenguaje.

Los programas escritos en MetaQuotes Language 5 tienen distintas propiedades y finalidades:

- **El Asesor Experto** es un sistema automático de comercio (SAC) que tiene como referencia un gráfico determinado. El Asesor Experto se inicia si surge un evento que éste puede procesar: eventos de inicialización y deinicialización, evento de recepción de un nuevo tick, evento procedente del temporizador, evento relacionado con el cambio de la profundidad de mercado, eventos procedentes del gráfico y los eventos del usuario. El Asesor Experto no sólo puede funcionar en modo de informar sobre la posibilidad de llevar a cabo una transacción, sino celebrar las transacciones en la cuenta comercial de una forma automatizada, enviándolas Inmediatamente al servidor comercial. Los Asesores Expertos se guardan en la carpeta con la siguiente dirección - *fichero_terminal\MQL5\Experts. La quintaesencia integral comunicante*
- **El indicador** personalizado es un indicador técnico creado personalmente por el usuario como complemento de los indicadores ya integrados en el terminal de cliente. Los indicadores personalizados, igual que los indicadores built-in, no pueden tradear de una forma automática y están destinados para realizar únicamente las funciones analíticas.
  Los indicadores personalizados se guardan en el directorio - *fichero_terminal\MQL5\Indicators*
- **El script** es un programa destinado para ejecutar alguna acción sólo una vez. A diferencia de los expertos, los scripts no procesan ningún evento salvo el de inicialización, para ello el script debe disponer de la función procesal OnStart. Los scripts se guardan en el directorio - *fichero_terminal \MQL5\Scripts*
- **La biblioteca** es un fichero de las funciones del usuario que está destinado para guardar y distribuir las partes de los programas del usuario de uso frecuente. Las bibliotecas no pueden activarse de una forma autónoma. Las bibliotecas se almacenan en el directorio - *fichero_terminal\MQL5\ Libraries*
- **El archivo de inserción** es un texto fuente de los bloques de los programas de usuario de uso frecuente. Durante la fase de compilación, estos archivos pueden ser incluidos en los textos fuentes de los expertos, scripts, indicadores personalizados y bibliotecas. Es más preferible utilizar los

archivos de inserción en vez de las bibliotecas debido a los gastos accesorios adicionales a la hora de recurrir a las funciones de la biblioteca.
Los archivos de inserción pueden estar ubicados en el mismo directorio donde se encuentra el archivo inicial. En este caso se utiliza la orden #include con las comillas dobles. Otro sitio para guardar los archivos de inserción es el directorio *fichero_terminal\MQL5\ Include*, utilizando en este caso la orden #include con los paréntesis angulares.

Bases del lenguaje

El lenguaje MetaQuotes Language 5 (MQL5) es un lenguaje de programación orientado a objetos de alto nivel y está destinado para diseñar las estrategias automáticas de trading, indicadores técnicos personalizados con el fin de analizar diferentes mercados financieros. No sólo permite diseñar diferentes sistemas expertos destinados para trabajar en tiempo real sino crear sus propias herramientas gráficas que ayudan a tomar decisiones comerciales.

MQL5 se basa en concepto del lenguaje de programación muy usual C++. En comparación con MQL4, han sido agregadas las enumeraciones, estructuras, clases y procesamiento de eventos. La interacción de los programas procesados en MQL5 con las demás aplicaciones mediante dll ha sido facilitado al máximo gracias al aumento de cantidad de tipos principales incorporados. La sintaxis del lenguaje MQL5 es parecida a la del C++, lo que permite traspasar a él sin ninguna dificultad los programas escritos en otros modernos lenguajes de programación.

Con el fin de aprender el lenguaje MQL5 todos los temas han sido agrupados en los siguientes apartados:

- Sintaxis
- Tipos de datos
- Operaciones y expresiones
- Operadores
- Funciones
- Variables
- Preprocesador
- Programación orientada a objetos

Sintaxis

En aspecto de la sintaxis el lenguaje de programación de estrategias comerciales MQL5 es muy parecido al lenguaje de programación C++, salvo algunas posibilidades:

- falta la aritmética de dirección;
- falta el operador goto;
- no se puede utilizar la enumeración anónima;
- no hay herencia múltiple.

Véase también

Enumeraciones, Estructuras y clases, Herencia

Comentarios

Los comentarios en bloque se empiezan con los símbolos /* y se terminan con */. Estos comentarios no pueden ser insertados. Los comentarios en línea se empiezan con los símbolos //, se terminan con el símbolo de nueva línea, se puede insertarlos en los comentarios en bloque. Esta permitido el uso de los comentarios en cualquier lugar donde se puede utilizar los espacios, además la cantidad de espacios no está limitada.

Ejemplos:

```
//--- Comentario en línea
/*  Comentario
    en          // Comentario en línea insertado
    bloque
*/
```

Identificadores

Los identificadores se utilizan como los nombres para las variables y funciones. Los identificadores no pueden tener más de 63 caracteres.

Los símbolos permitidos para escribir los identificadores son los siguientes: dígitos numéricos de 0 a 9, letras mayúsculas y minúsculas latinas a-z y A-Z se reconocen como símbolos distintos, guión bajo (_). Dígitos numéricos en ningún caso deben aparecer como primer carácter.

Un identificador no puede coincidir con una palabra reservada.

Ejemplos:

```
NAME1 name1 Total_5 Paper
```

Véase también

Variables, Funciones

Palabras reservadas

Los identificadores especificados más abajo se determinan como palabras reservadas a cada una de las cuales le corresponde una acción determinada, y no pueden ser utilizados en otro sentido:

Tipos de datos

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

Especificadores de acceso

const	private	protected
public	virtual	

Clases de memoria

extern	input	static

Operadores

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

Otros

false	#define	#property
this	#import	template
true	#include	typename

Tipos de datos

Cualquier programa opera con los los datos. Dependiendo de su función, éstos pueden ser de varios tipos. Por ejemplo, para acceder a los elementos de un array se utilizan los datos del tipo de números enteros. Los datos de precios tienen el tipo de doble precisión en punto flotante. Esto se debe a que en el lenguaje MQL5 no está previsto ningún tipo especial para los datos de precios.

Los datos de diferentes tipos se procesan a diferentes velocidades. El procesamiento de datos de números enteros se realiza más rápido. Para procesar datos de doble precisión se utiliza un coprocesador especial. No obstante, a causa de la complejidad de demostración interna de los datos de punto flotante, éstos requieren más tiempo para ser procesados que los de números enteros.

Los que más lentamente se procesan son los datos en cadena de caracteres. Esto está relacionado con la distribución y redistribución dinámica de la memoria operativa del ordenador.

Principales tipos de datos:

- enteros (char, short, int, long, uchar, ushort, uint, ulong)
- lógicos (bool)
- literales (char, uchar)
- cadenas de caracteres (string)
- de punto flotante (double, float)
- color (color)
- fecha y tiempo (datetime)
- enumeraciones (enum)

Tipos de datos compuestos:

- estructuras;
- clases.

En los términos de POO los tipos complejos de datos obtienen el nombre de los tipos abstractos de datos.

Los tipos color y datetime sirven unicamente para la comodidad de presentación e introducción de los parámetros hechos desde fuera: la tabla de propiedades del asesor o indicador del usuario (pestaña "Inputs"). Los datos de los tipos color y datetime se representan como los números enteros. Los tipos enteros y los del punto flotante se llaman los tipos aritméticos (numéricos).

En las expresiones se utiliza la conversión implícita de tipos si no se indica lo contario (la explícita).

Véase también

Conversión de tipos

Tipos enteros

En el lenguaje MQL5 los tipos enteros están representados por once categorías. Algunos de ellos se puede utilizar en conjunto con los demás si lo requiere la lógica del programa, aunque en este caso hay que tener en cuenta las normas de conversión de tipos.

En la tabla de abajo se muestran las características de cada tipo. Además, en la última columna, para cada tipo se indica su análogo en el lenguaje de programación C++.

Tipo	Tamaño en bytes	Valor mínimo	Valor máximo	Análogo en el lenguaje C++
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

Los valores de tipos enteros también pueden ser representados en forma de las constantes numéricas, literales de color, literales de fecha-hora, constantes de signos y enumeraciones.

Véase también

Conversión de datos, Constantes de tipos numéricos

Tipos char, short, int y long

char

El tipo entero char ocupa en la memoria 1 byte (8 bits) y permite representar en el sistema numérico binario 2^8 valores = 256. El tipo char puede contener los valores positivos, igual que negativos. El rango de valores es de -128 a 127.

uchar

El tipo entero uchar también ocupa en la memoria 1 byte, igual que el tipo char, pero a diferencia de él, uchar es destinado unicamente a los valores positivos. El valor mínimo es igual a cero, el valor máximo es igual a 255. La primera letra u del nombre uchar es la abreviatura de la palabra unsigned (sin signo).

short

El tipo entero short tiene el tamaño de 2 bytes (16 bits), permite representar la multitud de valores igual a 2 elevado a 16: 2^16 = 65 536. Puesto que el tipo short es con signos y contiene los valores tanto positivos, como negativos, el rango de valores se oscila entre -32 768 y 32 767.

ushort

El tipo ushort es el tipo short sin signos, también tiene el tamaño de 2 bytes. El valor mínimo es igual a cero, el valor máximo es igual a 65 535.

int

El tipo entero int tiene el tamaño de 4 bytes (32 bits). El valor mínimo es de -2 147 483 648, el valor máximo es de 2 147 483 647.

uint

El tipo entero sin signos uint ocupa en la memoria 4 bytes y permite representar los valores de números enteros de 0 a 4 294 967 295.

long

El tipo entero long tiene el tamaño de 8 bytes (64 bits). El valor mínimo es de -9 223 372 036 854 775 808, el valor máximo es de 9 223 372 036 854 775 807.

ulong

El tipo entero ulong también ocupa 8 bytes y permite almacenar valores de 0 a 18 446 744 073 709 551 615.

Ejemplos:

```
char  ch=12;
```

```
short sh=-5000;
int   in=2445777;
```

Debido a que los tipos enteros sin signos no sirven para almacenar los valores negativos, el intento de poner los valores negativos puede llevar a las consecuencias inesperadas. Pues un script inocente como éste llevará a un ciclo contínuo:

```
//--- ciclo continuo
void OnStart()
  {
   uchar  u_ch;

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

Así es correcto:

```
//--- versión correcta
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;
     }
  }
```

Resultado:

```
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
 ...
```

Ejemplos:

```
//--- no se puede guardar los valores negativos en los tipos sin signos
uchar  u_ch=-120;
ushort u_sh=-5000;
uint   u_in=-401280;
```

Hexadecimales: cífras 0-9, letras a-f o A-F para los valores 10-15, se empiezan con 0x o 0X.

Ejemplos:

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

Véase también

Conversión de tipos

Constantes de caracteres

En MQL5 los caracteres, como elementos de una línea, son los índices en conjunto de caracteres Unicode. Estos son unos valores hexadecimales que pueden ser convertidos en números enteros y con los que se puede ejecutar operaciones de números enteros, tales como, la suma y la resta.

Cualquier carácter solitario que viene entre comillas simples o el código ASCII hexadecimal en forma de '\x10' es una constante de caracteres y tiene el tipo ushort. Por ejemplo, la introducción del '0' representa el valor numérico 30 que corresponde al índice según el cual en la tabla de caracteres se encuentra el signo cero.

Ejemplo:

```
void OnStart()
  {
//--- determinemos las constantes de caracteres
   int symbol_0='0';
   int symbol_9=symbol_0+9; // obtenemos el signo '9'
//--- saquemos los valores de las constantes
   printf("En forma decimal: symbol_0 = %d,  symbol_9 = %d",symbol_0,symbol_9);
   printf("En forma hexadecimal: symbol_0 = 0x%x,  symbol_9 = 0x%x",symbol_0,symbol_9
//--- introduzcamos las constantes en una línea
   string test="";
   StringSetCharacter(test,0,symbol_0);
   StringSetCharacter(test,1,symbol_9);
//--- así se ven en la línea
   Print(test);
  }
```

Para un compilador durante el análisis de las líneas de constantes y las constantes de caracteres en el código fuente del programa la barra inversa es un carácter controlador. Algunos símbolos, por ejemplo, comillas simples ('), comillas dobles ("), barra inversa (\) y caracteres controladores se puede representar mediante una combinación de signos que se empiezan con la barra inversa (\), según se indica en la tabla de abajo:

Nombre del signo	Código mnemotécnico o imagen	Inscripción en MQL5	Valor numérico
nueva línea (cambio de línea)	LF	'\n'	13
tabulación horizontal	HT	'\t'	9
retorno de carro	CR	'\r'	10
barra inversa	\	'\\'	92
comilla simple	'	'\''	39
comilla doble	"	'\"'	34
código hexadecimal	hhhh	'\xhhhh'	de 1 a 4 símbolos hexadecimales
código decimal	d	'\d'	número decimal de 0 a 65535

Si después de la barra inversa sigue un signo distinto a los arriba mencionados, entonces el resultado no está determinado.

Ejemplo:

```
void OnStart()
  {
//--- declaremos las constantes de caracteres
   int a='A';
   int b='$';
   int c='©';      // código 0xA9
   int d='\xAE';   // código del signo ®
//--- imprimamos las constantes
   Print(a,b,c,d);
//--- agreguemos el signo a la línea
   string test="";
   StringSetCharacter(test,0,a);
   Print(test);
//--- cambiemos el signo en la línea
   StringSetCharacter(test,0,b);
   Print(test);
//--- cambiemos el signo en la línea
   StringSetCharacter(test,0,c);
   Print(test);
//--- cambiemos el signo en la línea
   StringSetCharacter(test,0,d);
   Print(test);
//--- representemos signos en forma numérica
   int a1=65;
   int b1=36;
   int c1=169;
   int d1=174;
//--- agreguemos el signo a la línea
   StringSetCharacter(test,1,a1);
   Print(test);
//--- cambiemos el signo en la línea
   StringSetCharacter(test,1,b1);
   Print(test);
//--- cambiemos el signo en la línea
   StringSetCharacter(test,1,c1);
   Print(test);
//--- cambiemos el signo en la línea
   StringSetCharacter(test,1,d1);
   Print(test);
  }
```

Como se decía antes, el valor de una constante de caracteres (o una variable) representa un índice en la tabla de signos, y debido a que éste es un número entero, se permite escribirlo de varias maneras.

```
void OnStart()
  {
//---
   int a=0xAE;      // código del signo ® corresponde al literal '\xAE'
   int b=0x24;      // código del signo $ corresponde al literal '\x24'
   int c=0xA9;      // código del signo © corresponde al literal '\xA9'
   int d=0x263A;    // código del signo O corresponde al literal '\x263A'
//--- saquemos los valores
   Print(a,b,c,d);
//--- agreguemos el signo a la línea
   string test="";
   StringSetCharacter(test,0,a);
   Print(test);
//--- cambiemos el signo en la línea
   StringSetCharacter(test,0,b);
   Print(test);
//--- cambiemos el signo en la línea
   StringSetCharacter(test,0,c);
   Print(test);
//--- cambiemos el signo en la línea
   StringSetCharacter(test,0,d);
   Print(test);
//--- códigos de palos
   int a1=0x2660;
   int b1=0x2661;
   int c1=0x2662;
   int d1=0x2663;
//--- agreguemos el signo de pica
   StringSetCharacter(test,1,a1);
   Print(test);
//--- agreguemos el signo de corazón
   StringSetCharacter(test,2,b1);
   Print(test);
//--- agreguemos el signo de diamante
   StringSetCharacter(test,3,c1);
   Print(test);
//--- agreguemos el signo de trébol
   StringSetCharacter(test,4,d1);
   Print(test);
//--- Ejemplo de literales de signos en la línea
   test="Dama\x2660As\x2662";
   printf("%s",test);
  }
```

La representación interna del literal de signo es el tipo ushort. Las constantes de caracteres pueden adquirir valores de 0 a 65535.

Véase también

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

Tipo datetime

El tipo datetime sirve para almacenar la fecha y la hora en forma de cantidad de segundos que han pasado desde el 1 de enero de 1970. Ocupa en la memoria 8 bytes.

Las constantes de fecha y hora pueden estar representadas por una línea de caracteres compuesta de 6 partes que representan el valor numérico del año, mes, día (o día, mes, año), hora, minuto y segundo. La constante se mete entre comillas simples y se empieza con el signo D.

El rango de valores es desde el 1 de enero de 1970 hasta el 31 de diciembre de 3000. Se puede omitir o la fecha (año, mes, día) o la hora (hora, minuto, segundo), o pueden ser las dos cosas.

Durante la especificación literal de la fecha, es deseable indicar el año, el mes y el día. Si no, el compilador mostrará un aviso sobre la entrada literal no completa.

Ejemplos:

```
datetime NY=D'2015.01.01 00:00';      // hora de inicio del año 2015
datetime d1=D'1980.07.19 12:30:27';   // año mes día horas minutos segundos
datetime d2=D'19.07.1980 12:30:27';   // igual a D'1980.07.19 12:30:27';
datetime d3=D'19.07.1980 12';         // igual a D'1980.07.19 12:00:00'
datetime d4=D'01.01.2004';            // igual a D'01.01.2004 00:00:00'
datetime compilation_date=__DATE__;             // fecha de compilación
datetime compilation_date_time=__DATETIME__;    // fecha y hora de compilación
datetime compilation_time=__DATETIME__-__DATE__;// hora de compilación
//--- ejemplos de declaraciones que provocarán los avisos del compilador
datetime warning1=D'12:30:27';        // igual a D'[fecha de compilación] 12:30:27'
datetime warning2=D'';                // igual a __DATETIME__
```

Véase también

Estructura de fecha, Fecha y hora, TimeToString, StringToTime

Tipo color

El tipo color sirve para almacenar la información sobre el color y ocupa en la memoria 4 bytes. El primer byte no se cuenta, los demás 3 bytes contienen los componentes RGB.

Las constantes de colores pueden ser representadas de tres formas distintas: de forma literal, con números enteros o mediante un nombre (sólo para los colores-Web concretos).

La representación literal se compone de tres partes que representan valores numéricos de la intensidad de tres componentes principales: rojo (red), verde (green), azul (blue). La constante se mete entre comillas simples y se empieza con el signo C. Los valores numéricos de la intensidad del componente de color se encuentra entre 0 y 255.

La representación con números enteros se realiza a través de un número decimal o hexadecimal. El número hexadecimal tiene la siguiente forma 0x00BBGGRR, donde RR es la intensidad del color rojo, GG - verde y BB - azul. Las constantes decimales no tienen la expresión directa en RGB. Estas representan el valor decimal de la expresión hexadecimal de números enteros.

Los colores concretos expresan un conjunto de colores-Web.

Ejemplos:

```
//--- literales
C'128,128,128'    // gris
C'0x00,0x00,0xFF' // azul
//nombres de colores
clrRed               // rojo
clrYellow            // amarillo
clrBlack             // negro
//--- representaciones con números enteros
0xFFFFFF          // blanco
16777215          // blanco
0x008000          // verde
32768             // verde
```

Véase también

Colores Web, ColorToString, StringToColor, Conversión de tipos

Tipo bool

El tipo bool sirve para almacenar los valores lógicos true (verdadero) o false (falso) la representacion de los cuales es 1 o 0 respectivamente.

Ejemplos:

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

La representación interna es un número entero de tamano de 1 byte. Cabe señalar que en las expresiones lógicas se puede utilizar en vez del tipo bool también otros tipos enteros o reales o representaciones de estos tipos, el compilador igual no va a señalar ningun error. En este caso el valor cero va a ser interpretado como false y los demás como true.

Ejemplos:

```
   int i=5;
   double d=-2.5;
   if(i) Print("i = ",i," y tiene el valor true");
   else Print("i = ",i,"y tiene el valor false");

   if(d) Print("d = ",d," y tiene el valor true");
   else Print("d = ",d," y tiene el valor false");

   i=0;
   if(i) Print("i = ",i," y tiene el valor true");
   else Print("i = ",i," y tiene el valor false");

   d=0.0;
   if(d) Print("d = ",d," y tiene el valor true");
   else Print("d = ",d," y tiene el valor false");

//--- resultados de ejecución
//   i= 5 y tiene el valor true
//   d= -2.5 y tiene el valor true
//   i= 0 y tiene el valor false
//   d= 0 y tiene el valor false
```

Véase también

Operaciones lógicas, Prioridades y orden de las operaciones

Enumeraciones

Los datos del tipo enumerativo enum se refieren a una cantidad limitada de datos. La definición del tipo enumerativo:

```
enum nombre del tipo enumerativo
  {
   lista de valores
  };
```

La lista de valores es una lista de variables separadas por comas.

Ejemplo:

```
enum months  // enumeración de las constantes concretas
   {
    January,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
   };
```

Después de declarar una enumeración aparece un nuevo tipo de datos de números enteros de 4 bytes. La declaración del nuevo tipo de datos permite al compilador controlar de una forma estricta los parámetros transferidos porque la enumeración establece nuevas constantes concretas. En el ejemplo de arriba la constante concreta January tiene el valor 0, February tiene el valor 1, December tiene el valor 11.

Regla: si una constante concreta, siendo un elemento de enumeración, no tiene adjudicado un valor concreto entonces su valor se forma de una manera automática. Si se trata del primer elemento de enumeración se le va a adjudicar el valor 0. Para todos los elementos posteriores los valores se calculan a base del valor del elemento anterior sumándole un uno.

Ejemplo:

```
enum intervals  // enumeración de las constantes concretas
   {
    month=1,     // intervalo de un mes
    two_months,  // dos meses
    quarter,     // tres meses - trimestre
    halfyear=6,  // semestre
    year=12,     // año - 12 meses
   };
```

Observaciones

- A diferencia de C++ el tamaño interno de la representación de un tipo enumerativo en MQL5 siempre es de 4 bytes. Es decir, sizeof(months) devolverá el valor 4.
- A diferencia de C++ en MQL5 no se puede declarar una enumeración anónima. Es decir, después de la palabra clave enum siempre hay que indicar un nombre único.

Véase también

Conversión de tipos

Tipos reales (double, float)

Los tipos reales (o tipos de punto flotante) representan valores que contienen la parte fraccionaria. En el lenguaje MQL5 existen dos tipos para los números con punto flotante. El modo de representar los números reales en la memoria del ordenador se rige por el estándar IEEE 754 y no depende de las plataformas, sistemas operativos y lenguajes de programación.

Tipo	Tamaño en bytes	Valor mínimo positivo	Valor máximo	Precisión de representación	Análogo en el lenguaje C++
float	4	1.175494351 e-38	3.402823466 e+38	7 cifras significativas	float
double	8	2.225073858 5072014e-308	1.797693134 8623158e +308	15 cifras significativas	double

El nombre double significa que la precisión de estos números es dos veces más que la precisión de los números del tipo float. En mayoría de los casos el tipo double es más cómodo. En muchos casos la precisión limitada de los números float simplemente es insuficiente. La razón de utilizar todavía el tipo float se encuentra en el ahorro de la memoria durante el almacenamiento (es importante para las grandes cantidades de matrices de números reales).

Las constantes de punto flotante se componen de la parte entera, punto (.) y parte fraccionaria. La parte entera y fraccionaria es una sucesión de números decimales.

Ejemplos:

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

Existe el modo científico de escribir las constantes reales. A menudo esta modo de escribir es más compacto en comparación con la forma tradicional.

Ejemplo:

```
double c1=1.12123515e-25;
double c2=0.000000000000000000000000112123515; // 24 ceros despues del punto decim

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

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

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

Hay que recordar que los números reales se almacenan en la memoria del ordenador con una cierta

precisión limitada en el sistema binario, mientras que el sistema decimal es de uso general. Por eso muchos números que se escriben en el sistema decimal, en el sistema binario pueden ser escritos sólo en forma de fracción contínua.

Por ejemplo, en el ordenador los números 0.3 y 0.7 están representados por las fracciones contínuas, mientras que el número 0.25 se guarda de forma exacta porque es la potencia de 2.

Por esta razón no se recomienda de ninguna manera comparar la igualdad de dos números reales porque esta comparación no es correcta.

Ejemplo:

```
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");
// Resultado: 1/3 + 4/3 != 5/3
  }
```

Si en alguna ocasión es necesario comparar la igualdad de dos números reales, entonces se puede hacerlo de dos maneras distintas. El primer modo consiste en la comparación de diferencia entre dos números con un valor pequeño que marca la precisión de comparación.

Ejemplo:

```
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",f_val);
   else Print("Different: d_val = ",DoubleToString(d_val,16),
              "  f_val = ",DoubleToString(f_val,16));
// Resultado: Different: d_val= 0.7000000000000000   f_val= 0.6999999880790710
  }
```

Cabe mencionar que el valor del parámetro epsilon en el ejemplo de arriba, no puede ser menos que la

constante predeterminada DBL_EPSILON. El valor de esta constante es 2.2204460492503131e-016. Para el tipo float la constante correspondiente es FLT_EPSILON = 1.192092896e-07. El sentido de estos valores es siguiente: se trata del valor mínimo que satisface la condición 1.0+DBL_EPSILON != 1.0 (para los números del tipo float 1.0+FLT_EPSILON != 1.0).

El segundo modo supone la comparación de la diferencia normalizada de dos números reales con el valor cero. Es inutil comparar la diferencia de los números normalizados con el cero porque el resultado de cualquier operación matemática con los números normalizados no va a ser normalizado.

Ejemplo:

```
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));
// Resultado: Different: d_val= 0.3000000000000000   f_val= 0.3000000119209290
  }
```

Como resultado de algunas operaciones matemáticas del coprocesador se puede obtener un número real extendido, el que no se puede utilizar en las operaciones matemáticas y operaciones de comparación porque el resultado de ejecución de las operaciones con números reales extendidos no está definido. Por ejemplo, tratando de calcular el arcseno de 2 se obtiene el infinito negativo.

Ejemplo:

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

A parte del infinito negativo existe el infinito positivo y NaN (no es un número). Para determinar que el número es extendido se puede utilizar la función MathIsValidNumber(). Según el estándar IEEE ellos tienen una representación informática especial. Por ejemplo, el infinito positivo para el tipo double tiene la representación de bit 0x7FF0 0000 0000 0000.

Ejemplos:

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

```
//--- empecemos
   str1 s1;
   str2 s2;
//---
   s1.d=MathArcsin(2.0);         // obtenemos número extendido -1.#IND
   s2=s1;
   printf("1.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0xFFFF000000000000;      // número extendido -1.#QNAN
   s1=s2;
   printf("2.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF7000000000000;      // nonúmero máximo SNaN
   s1=s2;
   printf("3.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF8000000000000;      // nonúmero mínimo QNaN
   s1=s2;
   printf("4.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FFF000000000000;      // nonúmero máximo QNaN
   s1=s2;
   printf("5.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0x7FF0000000000000;      // infinito positivo 1.#INF y nonúmero mínimo SNaN
   s1=s2;
   printf("6.   %f %I64X",s1.d,s2.l);
//---
   s2.l=0xFFF0000000000000;      // infinito negativo -1.#INF
   s1=s2;
   printf("7.  %f %I64X",s1.d,s2.l);
//---
   s2.l=0x8000000000000000;      // cero negativo -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;      // número máximo normalizado (MAX_DBL)
   s1=s2;
   printf("11.  %.16e %I64X",s1.d,s2.l);
```

```
//---
   s2.l=0x0010000000000000;      // mínimo normalizado positivo (MIN_DBL)
   s1=s2;
   printf("12.  %.16e %.16I64X",s1.d,s2.l);
//---
   s1.d=0.7;                     // indicamos que el número 0.7 - es una fracción cont
   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
*/
```

Véase también

DoubleToString, NormalizeDouble, Constantes de tipos numéricos

Tipo string

Tipo string sirve para guardar las cadenas de caracteres. Una cadena de caracteres es una sucesión de caracteres en formato Unicode con el cero al final. A una variable string se le puede asignar una constante literal. Una constante literal es una sucesión de caracteres Unicode encerada entre comillas dobles: "Es una constante literal".

Para poder introducir una comilla doble (") dentro de la cadena hay que interponerle el signo de barra inversa (\). Cualquier constante de signo especial, si le interviene el signo de barra inversa (\), puede ser introducida en la cadena.

Ejemplos:

```
string svar="This is a character string";
string svar2=StringSubstr(svar,0,4);
Print("Símbolo de copyright\t\x00A9");
FileWrite(handle,"esta cadena contiene el símbolo de avance de línea \n");
string MT5path="C:\\Program Files\\MetaTrader 5";
```

Para hacer el código fuente más cómodo para leer, las cadenas constantes largas se puede dividir en partes sin la operación de adición. Durante el proceso de compilación, todas estas partes se unirán en una cadena larga:

```
//--- declararemos una cadena constante larga
   string HTML_head="<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"
                    " \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">\n"
                    "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n"
                    "<head>\n"
                    "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=u
                    "<title>Trade Operations Report</title>\n"
                    "</head>";
//--- mostraremos la cadena constante en el diario
   Print(HTML_head);
  }
```

Véase también

Conversión de tipos, Funciones de cadenas de caracteres, FileOpen, FileReadString, FileWriteString

Estructuras y clases

Estructuras

Una estructura es un conjunto de elementos del tipo libre, salvo el tipo void. De esa manera, la estructura une los datos de diferentes tipos que están vinculados de forma lógica.

Declaración de estructura

El tipo de datos estructural se define de la siguiente manera:

```
struct nombre_de_estructura
  {
   descripción_de_elementos
  };
```

No se puede usar el nombre de la estructura en calidad del identificador (nombre de la variable o función). Hay que tener en cuenta que en MQL5 los elementos de una estructura siguen directamente uno detrás del otro sin que sean alineados. En el lenguaje C++ este comando se proporciona al compilador mediante la instrucción

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

Si hace falta hacer otra alineación dentro de la estructura, es necesario utilizar los elementos "de relleno" adicionales de tamaño necesario.

Ejemplo:

```
struct trade_settings
  {
   uchar  slippage;     // valor del deslizamiento permitido - tamaño 1 byte
   char   reserved1;    // 1 byte de permiso
   short  reserved2;    // 2 bytes de permiso
   int    reserved4;    // otros 4 bytes de permiso. Aseguramos la alineación al marg
   double take;         // valor del precio de fijación del beneficio
   double stop;         // valor del precio del stop de protección
  };
```

Esta descripción de alineación de las estructuras es necesaria unicamente para la transmisión a las funciones dll importadas.

Atención: este ejemplo refleja los datos proyectados de una manera errónea. Sería mejor declarar al principio los datos take y stop de mayor tamaño que el tipo double, y luego declarar el elemento slippage del tipo uchar. En este caso, la presentación interna de los datos siempre va a ser igual independientemente del valor indicado en #pragma pack().

Si la estructura contiene las variables del tipo string y/o el objeto del array dinámico, entonces para esta estructura el compilador asigna un constructor implícito donde se efectúa la anulación de todos los elementos del tipo string y la inicialización correcta para el objeto del array dinámico.

Estructuras simples

Las estructuras que no contienen las cadenas y objetos del array dinámico se llaman estructuras simples, las variables de estas estructuras pueden copiarse libremente una en otra, incluso si se trata de las estructuras diferentes. Las variables de estructuras simples, igual que sus matrices, pueden ser pasadas como parámetros en las funciones importadas de DLL.

Acceso a los elementos de la estructura

El nombre de la estructura es un tipo de datos nuevo y permite declarar las variables de este tipo. Se puede declarar la estructura sólo una vez dentro de un proyecto. El acceso a los elementos de las estructuras se realiza mediante operación punto (.).

Ejemplo:

```
struct trade_settings
  {
   double take;         // valor del precio de fijación del beneficio
   double stop;         // valor del precio del stop de protección
   uchar  slippage;     // valor del deslizamiento permitido
  };
//--- creamos e inicializamos la variable del tipo trade_settings
trade_settings my_set={0.0,0.0,5};
if (input_TP>0) my_set.take=input_TP;
```

Clases

Las clases llevan una serie de diferencia de las estructuras:

- en la declaración se utiliza la palabra clave class;
- si no se indica lo contrario todos los elementos de las clase por defecto tienen el especificador de acceso private. Los elementos-datos de la estructura por defecto tienen el tipo de acceso public, si no se indica lo contrario;
- los objetos de las clases siempre tienen una tabla de funciones virtuales, incluso si en la clase ninguna función virtual esté declarada. Las estructuras no pueden tener funciones virtuales;
- las clases pueden ser heredadas unicamente de las clases, y las estructuras sólo de las estructuras.

Las clases y las estructuras pueden tener el constructor y destructor explícitos. En el caso, si el constructor está determinado de una manera explícita, la inicialización de variable del tipo de la estructura o clase con la ayuda de la sucesión inicializadora es imposible.

Ejemplo:

```
struct trade_settings
  {
   double take;         // valor del precio de fijación del beneficio
   double stop;         // valor del precio del stop de protección
   uchar  slippage;     // valor del deslizamiento permitido
   //--- constructor
          trade_settings() { take=0.0; stop=0.0; slippage=5; }
   //--- destructor
         ~trade_settings() { Print("Es el final"); }
  };
```

```
//--- compilador mostrará el error con el mensaje sobre la imposibilidad de inicializ
trade_settings my_set={0.0,0.0,5};
```

Constructores y destructores

El constructor es una función especial que se llama automáticamente cuando se crea un objeto de estructura o clase, y normalmente se utiliza para la inicialización de los miembros de la clase. A continuación vamos a hablar sólo de las clases, pero todo lo dicho también se refiere a las estructuras, si no se especifica lo otro. El nombre del constructor debe coincidir con el de la clase. El constructor no tiene el tipo devuelto (se puede indicar el tipo void).

Algunos miembros definidos de la clase, tales como - cadenas, arrays dinámicos y objetos que requieren la inicialización - de cualquier manera serán inicializados, independientemente de la presencia del constructor.

Cada clase puede tener varios constructores que se diferencian por el número de parámetros y listas de inicialización. Un constructor que se requiere la especificación de parámetros se llama el constructor paramétrico.

Un constructor que no tiene parámetros se llama un **constructor por defecto**. Si en la clase no está declarado ningún constructor, entonces durante la compilación el compilador creará un constructor por defecto.

```
//+------------------------------------------------------------------+
//|  clase para trabajar con la fecha                                |
//+------------------------------------------------------------------+
class MyDateClass
  {
private:
   int               m_year;           // año
   int               m_month;          // mes
   int               m_day;            // día del mes
   int               m_hour;           // hora del día
   int               m_minute;         // minutos
   int               m_second;         // segundos
public:
   //--- constructor por defecto
                     MyDateClass(void);
   //--- constructor paramétrico
                     MyDateClass(int h,int m,int s);
  };
```

Se puede declarar el constructor en la descripción de la clase y luego definir su cuerpo. Por ejemplo, así se puede definir dos constructores de la clase MyDateClass:

```
//+------------------------------------------------------------------+
//| constructor por defecto                                          |
//+------------------------------------------------------------------+
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__);
  }
//+------------------------------------------------------------------+
//| constructor paramétrico                                          |
//+------------------------------------------------------------------+
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__);
  }
```

En el constructor por defecto se llenan todos los miembros de la clase por medio de la función TimeCurrent(), en el constructor paramétrico se llenan sólo los valores de la hora. Los demás miembros de la clase (m_year, m_month y m_day) serán inicializados automáticamente con la fecha en curso.

El constructor por defecto tiene un propósito especial cuando se inicializa un array de objetos de su clase. El constructor cuyos parámetros tienen los valores por defecto, **no es** constructor por defecto. Ejemplificaremos esto:

```
//+------------------------------------------------------------------+
//|  clase con un constructor por defecto                            |
//+------------------------------------------------------------------+
class CFoo
  {
   datetime          m_call_time;     // hora de la última llamada al objeto
public:
   //--- un constructor con el parámetro que tiene el valor predefinido no es un cons
                     CFoo(datetime t=0){m_call_time=t;};
   string ToString(){return(TimeToString(m_call_time,TIME_DATE|TIME_SECONDS));};
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
```

```
//+------------------------------------------------------------------+
void OnStart()
  {
   CFoo foo; // Esta opción se puede utilizar - se llamará a un constructor con parám
//--- posibles opciones de creación del objeto CFoo
   CFoo foo1(TimeCurrent());     // la primera opción de creación automática del obje
   CFoo foo2();                  // la segunda opción de creación automática del obje
   CFoo foo3=TimeTradeServer();  // la tercera opción de creación automática del obje
//--- posibles opciones de creación de punteros CFoo por medio del operador new
   CFoo *foo4=new CFoo();
   CFoo *foo5=new CFoo(TimeTradeServer());
   CFoo *foo6=GetPointer(foo5);  // ahora foo5 y foo6 apuntan al mismo objeto
   CFoo *foo7,*foo8;
   foo7=new CFoo(TimeCurrent());
   foo8=GetPointer(foo7);        // foo7 y foo8 apuntan al mismo objeto
   //CFoo foo_array[3];     // esta opción no se puede utilizar - el constructor por
   //CFoo foo_dyn_array[];  // esta opción no se puede utilizar - el constructor por
//--- mostramos los valores m_call_time
   Print("foo.m_call_time=",foo1.ToString());
   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("foo6.m_call_time=",foo6.ToString());
   Print("foo7.m_call_time=",foo7.ToString());
   Print("foo8.m_call_time=",foo8.ToString());
//--- eliminaremos los objetos creados dinámicamente
   delete foo4;
   delete foo5;
   delete foo7;
  }
```

Si añadimos comentarios a estas cadenas en este ejemplo

```
   //CFoo foo_array[3];     // esta opción no se puede utilizar - el constructor por d
```

o

```
   //CFoo foo_dyn_array[];  // esta opción no se puede utilizar - el constructor por d
```

el compilador devolverá el error para ellas "default constructor is not defined".

Si la clase tiene un constructor declarado por el usuario, entonces el compilador no generará el constructor por defecto. Esto quiere decir que si en una clase está declarado un constructor paramétrico pero no está declarado un constructor por defecto, entonces no se puede declarar los arrays de los objetos de esta clase. Pues, para este script el compilador devolverá el error:

```
//+------------------------------------------------------------------+
//| clase sin constructor por defecto                                |
//+------------------------------------------------------------------+
```

```
class CFoo
  {
   string            m_name;
public:
                     CFoo(string name) { m_name=name;}
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- durante la compilación obtenemos el error "default constructor is not defined"
   CFoo badFoo[5];
  }
```

En este ejemplo la clase CFoo tiene declarado un constructor paramétrico - en este caso durante la compilación el compilador no crea automáticamente el constructor por defecto. Al mismo tiempo, cuando se declara un array de objetos se supone que todos los objetos tienen que ser creados e inicializados automáticamente. Durante la inicialización automática del objeto, es necesario llamar a un constructor por defecto, pero debido a que el constructor por defecto no está declarado explícitamente y no ha sido generado automáticamente por el compilador, entonces resulta imposible crear este objeto. Precisamente por esta razón el compilador muestra el error aún en la fase de compilación.

Existe una sintaxis especial para la inicialización del objeto mediante el constructor. Los inicializadores del constructor (construcciones especiales para la inicialización) para los miembros de una estructura o clase se puede especificar en la lista de inicialización.

La lista de inicialización es una lista de inicializadores separados por comas que sigue tras dos puntos después de la lista de parámetros del constructor y precede el cuerpo (va antes de la llave que abre). Existen algunas exigencias:

- las listas de inicialización se puede utilizar sólo en los constructores;
- no se puede inicializar los miembros de los padres en la lista de inicialización;
- tras las lista de inicialización debe ir la definición (implementación) de la función.

Vamos a mostrar algunos ejemplos de constructores para la inicialización de los miembros de la clase.

```
//+------------------------------------------------------------------+
//| clase para almacenar los apellidos y el nombre de una persona
//+------------------------------------------------------------------+
class CPerson
  {
   string            m_first_name;     // nombre
   string            m_second_name;    // apellido
public:
   //--- constructor por defecto vacío
                     CPerson() {Print(__FUNCTION__);};
   //--- constructor paramétrico
                     CPerson(string full_name);
   //--- constructor con la lista de inicialización
```

```
                     CPerson(string surname,string name): m_second_name(surname), m_f
   void PrintName(){PrintFormat("Name=%s Surname=%s",m_first_name,m_second_name);};
  };
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
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()
  {
//--- obtenemos el 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("Whinnie","Pooh");   // Winnie the Pooh
   //--- mostraremos los valores
   Tom.PrintName();
   Huck.PrintName();
   Pooh.PrintName();

   //--- eliminaremos el objeto creado dinámicamente
   delete Pooh;
  }
```

En este caso, la clase CPerson tiene tres constructores:

1. un constructor por defecto explícito que permite crear un array de los objetos de esta clase;
2. un constructor con un parámetro que obtiene el nombre completo como parámetro, y lo divide en el nombre y el apellido según el espacio encontrado;
3. un constructor con dos parámetros que contiene la lista de inicialización. Los inicializadores - m_second_name(surname) y m_first_name(name).

Fíjese cómo la inicialización reemplazó la asignación utilizando la lista. Los miembros individuales deben inicializarse como sigue:

```
miembro_de_la_clase (lista de expresiones)
```

Los miembros pueden seguir cualquier orden en la lista de inicialización, pero todos los miembros de la clase van a inicializarse según el orden de su declaración. Esto significa que en el tercer constructor primero será inicializado el miembro m_first_name porque va declarado primero, y sólo después de él

será inicializado el miembro m_second_name. Esto hay que tener en cuenta cuando la inicialización de unos miembros de la clase depende de los valores en otros miembros de la clase.

Si en la clase base no está declarado el constructor por defecto pero al mismo tiempo está declarado uno o varios constructores paramétricos, habrá que llamar sí o sí a uno de los constructores de la clase base en la lista de inicialización. Éste va tras la coma, como los demás miembros de la lista, y será llamado en primer lugar durante la inicialización del objeto independientemente de su ubicación en la lista de inicialización.

```
//+------------------------------------------------------------------+
//|  clase base                                                      |
//+------------------------------------------------------------------+
class CFoo
  {
   string            m_name;
public:
   //--- constructor con la lista de inicialización
                     CFoo(string name) : m_name(name) { Print(m_name);}
  };
//+------------------------------------------------------------------+
//|  descendiente de la clase CFoo                                   |
//+------------------------------------------------------------------+
class CBar : CFoo
  {
   CFoo              m_member;      // el miembro de la clase es el objeto del padre
public:
   //--- el constructor por defecto en la lista de inicialización llama al constructo
                     CBar(): m_member(_Symbol), CFoo("CBAR") {Print(__FUNCTION__);}
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   CBar bar;
  }
```

En el ejemplo mencionado, durante la creación del objeto bar se llamará al constructor por defecto CBar() en el que primero se llama al constructor para el padre CFoo, y luego se llama al constructor para el miembro de la clase m_member.

Los destructores son unas funciones especiales llamadas automáticamente a la hora de eliminar un objeto de la clase. El nombre del destructor se escribe como el de la clase con la tilde (~). Las cadenas, arrays dinámicos y los objetos que necesitan deinicialización van a ser deinicializados de cualquier manera independientemente de la presencia del destructor. Disponiendo del destructor, estas acciones van a ser ejecutadas después del arranque del mismo.

Los destructores siempre son virtuales, independientemente de que si están declarados con la palabra clave virtual o no.

Determinación de los métodos de clase

Las funciones-métodos de clase pueden ser determinados tanto dentro de la clase, como fuera de la declaración de la clase. Si el método se determina dentro de la clase, entonces su cuerpo sigue directamente después de la declaración del método.

Ejemplo:

```
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(++m_turn>3) m
   virtual void      Draw()                     { return;                }
   virtual bool      CheckDown(int& pad_array[]);
   virtual bool      CheckLeft(int& side_row[]);
   virtual bool      CheckRight(int& side_row[]);
  };
```

Las funciones con SetRightBorder(int border) por Draw() se declaran y se determinan directamente dentro de la clase CTetrisShape.

El constructor CTetrisShape() y los métodos CheckDown(int& pad_array[]), CheckLeft(int& side_row[]) y CheckRight(int& side_row[]) se declaran sólo dentro de la clase, pero por ahora no están determinados. Las determinaciones de estas funciones deben seguir más adelante en el código. Para determinar el método fuera de la clase se utiliza la operación del permiso de contexto, como contexto se utiliza el nombre de la clase.

Ejemplo:

```
//+------------------------------------------------------------------+
//| Constructor de la clase base                                     |
//+------------------------------------------------------------------+
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;
  }
//+------------------------------------------------------------------+
//| Prueba de posibilidad de moverse abajo (para la vara o el cubo)  |
//+------------------------------------------------------------------+
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);
  }
```

Modificadores de acceso public, protected y private

A la hora de crear nueva clase se recomienda limitar el acceso a los elementos desde fuera. Para eso se utilizan las palabras claves private o protected. En este caso el acceso a los datos encubiertos puede realizarse sólo desde las funciones-métodos de la misma clase. Si se utiliza la palabra clave protected, entonces el acceso a los datos encubiertos se puede realizar también de los métodos de las clases que son herederos de esta clase. De la misma manera se puede limitar el acceso a las funciones-métodos de clase.

Si se necesita abrir totalmente el acceso a los elementos y/o métodos de clase, entonces se utiliza la palabra clave public.

Ejemplo:

```
class CTetrisField
  {
private:
   int               m_score;                          // cuenta
   int               m_ypos;                           // posición actual de la pie
   int               m_field[FIELD_HEIGHT][FIELD_WIDTH]; // matrix del vaso
   int               m_rows[FIELD_HEIGHT];             // numeración de las filas d
   int               m_last_row;                       // la última fila libre
   CTetrisShape     *m_shape;                          // pieza del tetris
   bool              m_bover;                          // fin del juego
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();
  };
```

Cualquier elemento y método de la clase que están declarados después del especificador public (y hasta el siguiente especificador del acceso), son accesibles a la hora de cualquier referencia del programa hacia el objeto de esta clase. En este ejemplo son los siguientes elementos: funciones CTetrisField(), Init(), Deinit(), Down(), Left(), Right(), Rotate() y Drop().

Cualquier elemento de la clase que está declarado después del especificador private (y hasta el siguiente especificador del acceso), es accesible sólo para las funciones-elementos de la misma clase. Los especificadores de acceso a los elementos siempre se terminan con dos puntos (:) y pueden aparecer en la determinación de la clase varias veces.

El acceso a los elementos de la clase base puede volver a determinarse durante la herencia en las clases derivadas.

Véase también

Programación orientada a objetos

Objeto de un array dinámico

Arrays dinámicos

Se permite declarar un array que no no supere 4 dimensiones. Al declarar una matriz dinámica (matriz sin valor expreso en el primer par de los corchetes), el compilador crea automáticamente una variable de la estructura indicada más arriba (objeto de matriz dinámica) y proporciona el código para una inicialización correcta.

Los arrays dinámicos se liberan automáticamente, al salir fuera de los límites de la visibilidad del bloque donde han sido declarados.

Ejemplo:

```
double matrix[][10][20]; // array dinámica de 3 dimensiones
ArrayResize(matrix,5);   // indicamos el tamaño de la primera dimensión
```

Arrays estáticos

Si todas las dimensiones significativas de un array se indican de una manera explícita, el compilador de antemano distribuye el espacio necesario de la memoria. Este array se llama array estático. No obstante, el compilador distribuye adicionalmente la memoria para el objeto del array dinámico. Este objeto está vinculado con el búfer estático distribuido previamente (sector de memoria para almacenamiento de la matriz).

La creación de un objeto del array dinámico está condicionado a la posible necesidad de transmitir dicho array estático como un parámetro a alguna función.

Ejemplos:

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

Arrays como partes de las estructuras

Al declarar un array estático como un elemento de una estructura, el objeto del array dinámico no se crea. Esto se hace para la compatibilidad de las estructuras de datos que se utilizan en Windows API.

Sin embargo, los arrays estáticos declarados como elementos de las estructuras, también se puede pasar a las funciones MQL5. En este caso, al pasar el parámetro va a crearse un objeto de un array dinámico temporal vinculado con el array estático - elemento de la estructura.

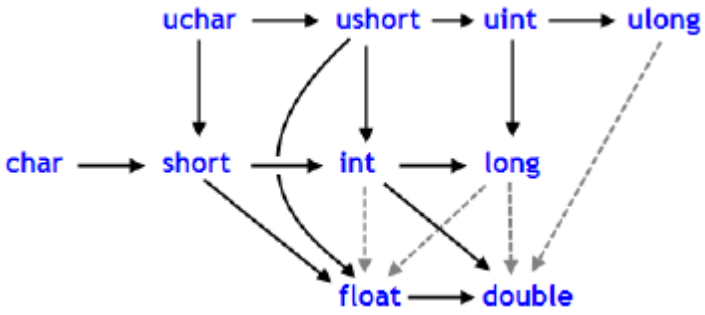
Véase también

Operaciones con arrays, Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Conversión de tipos

Transformación de tipos numéricos

Muy a menudo surge la necesidad de transformar un tipo de datos en otro. No cada uno de los tipos numéricos puede ser transformado en otro. Las transformaciones admisibles en MQL5 se indican en el esquema:



Las líneas contínuas con flechas indican las transformaciones que se efectuan sin pérdida de la información. El tipo bool puede figurar en vez del tipo char (ambos ocupan 1 byte de la memoria), en vez del tipo int se puede utilizar el tipo color (4 bytes cada uno), y en vez del tipo long se admite el tipo datetime (ocupan 8 bytes cada uno). Cuatro líneas rayadas con flechas de color gris significan las transformaciones en las que puede ocurrirse la pérdida de precisiones. Por ejemplo, la cantidad de dígitos en el número entero 123456789 (int) supera la cantidad de dígitos que puede ser representada en el tipo float.

```
   int n=123456789;
   float f=n;     // contenido f es igual a 1.234567892E8
   Print("n = ",n,"   f = ",f);
   // resultado n= 123456789    f= 123456792.00000
```

El número transformado en el tipo float tiene el mismo orden pero con una precisión algo menor. Las transformaciones inversas a las flechas negras se realizan con una posible pérdida de la información. Las transformaciones entre char y uchar, short y ushort, int y uint, long y ulong (se tienen en cuenta las transformaciones directas e inversas) pueden llevar a la pérdida de la información.

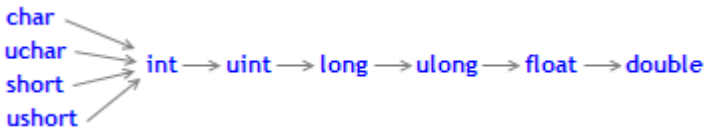
Resulta que al transformar los valores con punto flotante en el tipo entero, la parte fraccionaria siempre se omite. Si se necesita redondear el número con punto flotante hasta un número entero más próximo (lo que en mayoría de los casos puede ser más útil) es necesario utilizar la función MathRound ().

Ejemplo:

```
//--- aceleración de la gravedad
   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);
/*
   Resultado:
```

```
   round_g = 9
   math_round_g = 10
*/
```

Si dos valores se unen con un operador binario, entonces antes de iniciar la operación el operando del tipo menor se transforma en el tipo mayor según la prioridad indicada en la esquema:



En las operaciones los tipos de datos char, uchar, short y ushort se convierten al tipo int incondicionalmente.

Ejemplos:

```
   char   c1=3;
//--- primer ejemplo
   double d2=c1/2+0.3;
   Print("c1/2 + 0.3 = ",d2);
// Resultado:   c1/2+0.3 = 1.3

//--- segundo ejemplo
   d2=c1/2.0+0.3;
   Print("c1/2.0 + 0.3 = ",d2);
// Resultado:   c1/2.0+0.3 = 1.8
```

La expresión calculada se compone de dos partes. En el primer ejemplo la variable c1 del tipo char se transforma en la variable temporal del tipo int, puesto que el segundo operando en la operación de división, la constante 2, tiene el tipo int mayor. Como resultado de la división de números enteros 3/2 obtenemos el valor 1 que tiene el tipo int.

En la segunda operación del primer ejemplo el segundo operando es la constante 0.3 que tiene el tipo double, por eso el resultado de la primera operación se transforma en la variable temporal del tipo double con el valor 1.0.

En el segundo ejemplo la variable c1 del tipo char se transforma en la variable temporal del tipo double, puesto que el segundo operando en la operación de división, la constante 2.0, tiene el tipo double. Las siguientes transformaciones no se realizan.

La conversión de tipos numéricos

En las expresiones del lenguaje MQL5 se puede utilizar tanto la conversión explícita de tipos, como la implícita. La conversión explícita de tipos se escribe de la siguiente manera:

```
var_1 = (tipo)var_2;
```

En calidad de la variable var_2 puede ser una expresión o resultado de ejecución de la función. Se admite también la inscipción funcional de la conversión explícita de tipos:

```
var_1 = tipo(var_2);
```

Vamos a analizar la conversión explícita a base del primer ejemplo.

```
//--- tercer ejemplo
   double d2=(double)c1/2+0.3;
   Print("(double)c1/2 + 0.3 = ",d2);
// Resultado:   (double)c1/2+0.3 = 1.80000000
```

Antes de realizar la operación de división la variable c1 se convierte al tipo double. Ahora la constante de número entero 2 se convierte al valor 2.0 del tipo double, puesto que en el resultado de la transformación el primer operando ha obtenido el tipo double. Prácticamente la conversión explícita es una operación unaria.

Además, al intentar realizar la conversión de tipos, el resultado puede salir de los márgenes del rango permitido. En este caso habrá el acortamiento. Por ejemplo:

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

Antes de ejecutar las operaciones (salvo la de asignación), ocurre la transformación en el tipo que tenga la mayor prioridad, y antes de las operaciones de asignación – en el tipo objetivo.

Ejemplos:

```
   int    i=1/2;         // no hay conversión de tipos, resultado: 0
   Print("i = 1/2  ",i);

   int k=1/2.0;          // la expresión se convierte al tipo double,
   Print("k=1/2  ",k);   // luego se convierte al tipo objetivo int, resultado: 0

   double d=1.0/2.0;     // no hay conversión de tipos, resultado: 0.5
   Print("d=1/2.0; ",d);

   double e=1/2.0;       // la expresión se convierte al tipo double,
   Print("e=1/2.0; ",e);// que coincide con el tipo objetivo, resultado: 0.5

   double x=1/2;         // la expresión del tipo int se convierte al tipo objetivo do
   Print("x=1/2; ",x);   // resultado: 0.0
```

Durante la conversión del tipo long/ulong al tipo double puede que se pierda la precisión: si el valor entero es más de 9223372036854774784 o menos de -9223372036854774784.

```
void OnStart()
  {
   long l_max=LONG_MAX;
   long l_min=LONG_MIN+1;
//--- buscamos el valor máximo entero que no pierde la precisión cuando se convierte
   while(l_max!=long((double)l_max))
      l_max--;
```

```
//--- buscamos el valor mínimo entero que no pierde la precisión cuando se convierte
   while(l_min!=long((double)l_min))
      l_min++;
//--- ahora derivamos el intervalo encontrado para los números enteros
   PrintFormat("Si un número entero se convierte a double, tiene que "
               "encontrarse en el intervalo [%I64d, %I64d]",l_min,l_max);
//--- ahora vamos a ver qué es lo que pasa si el número se queda fuera de este interv
   PrintFormat("l_max+1=%I64d, double(l_max+1)=%.f, ulong(double(l_max+1))=%I64d",
               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))=%I64d",
               l_min-1,double(l_min-1),long(double(l_min-1)));
//--- el resultado será el siguiente
// Cuando un número entero se convierte a double, este número tiene que encontrarse d
// l_max+1=9223372036854774785, double(l_max+1)=9223372036854774800, ulong(double(l_m
// l_min-1=-9223372036854774785, double(l_min-1)=-9223372036854774800, ulong(double(l
  }
```

Las conversiones para el tipo string

El tipo string tiene la mayor prioridad entre los tipos simples. Por esta razón, si en la operación uno de los operandos tiene el tipo string, entonces el otro operando va a ser convertido al tipo string de una manera automática. Hay que tener en cuenta que para el tipo string se permite sólo operación binaria de suma. Está permitida la conversión explícita de la variable del tipo string a cualquier tipo numérico.

Ejemplos:

```
   string s1=1.0/8;            // la expresión se convierte al tipo double,
   Print("s1=1.0/8; ",s1);     // luego al tipo objetivo string,
// resultado:"0.12500000"(cadena de 10 símbolos)

   string s2=NULL;             // deinicialización de la cadena
   Print("s2=NULL; ",s2);      // resultado: cadena vacía
   string s3="Ticket N"+12345; // la expresión se convierte al tipo string
   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));
```

La conversión de datos de tipo de las estructuras simples

Los datos de tipo de las estructuras simples se puede asignar uno al otro sólo en el caso si todos los elementos de las ambas estructuras tienen los tipos numéricos. Entendiéndose que en las operaciones de asignación los dos operandos, tanto de la izquierda como de la derecha, deben ser del tipo de las estructuras. No se hace una conversión por elementos como tal, simplemente se copia. Si los tamaños de las estructuras se diferencian, entonces se copia la cantidad de bytes que corresponde al menor tamaño. De esta manera se compensa la falta de las uniones (union) en el lenguaje MQL5.

Ejemplos:

```
struct str1
  {
   double  d;
  };
//---
struct str2
  {
   long    l;
  };
//---
struct str3
  {
   int     low_part;
   int     high_part;
  };
//+------------------------------------------------------------------+
void OnStart()
  {
   str1 s1;
   str2 s2;
   str3 s3;
//---
   s1.d=MathArcsin(2.0);        // obtenemos el número irreal -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);
  }
```

Otro ejemplo nos demuestra como se puede organizar la propia función para obtener del tipo color las representaciones del color en RGB (Red,Green,Blue). Para eso vamos a crear dos estructuras que tengan el mismo tamaño pero diferentes contenidos. Para la comodidad vamos a añadir en la estructura una función que devuelva el color en representación de RGB en forma de la cadena.

```
#property script_show_inputs
input color          testColor=clrBlue;// asignen el color para su evaluación
//--- estructura para la presentación del color en RGB
struct RGB
  {
```

```
   uchar             blue;          // componente azul del color
   uchar             green;         // componente verde del color
   uchar             red;           // componente rojo del color
   uchar             empty;         // este byte no se usa
   string            toString();    // función para la obtención en forma de la caden
  };
//--- función para la salida en forma de la cadena
string RGB::toString(void)
  {
   string out="("+(string)red+":"+(string)green+":"+(string)blue+")";
   return out;
  }
//--- estructura para almacenamiento del tipo interno color
struct builtColor
  {
   color             c;
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- variable para almacenamiento en RGB
   RGB colorRGB;
//--- variable para almacenamiento del tipo color
   builtColor test;
   test.c=testColor;
//--- conversión de dos estructuras mediante la copia del contenido
   colorRGB=test;
   Print("color ",test.c,"=",colorRGB.toString());
//---
  }
```

La conversión de tipos de punteros de las clases bases a los punteros de las clases derivadas

Los objetos de la clase derivada abiertamente también pueden considerarse como los objetos de la clase base que le corresponde. Esto lleva a algunos efectos interesantes. Por ejemplo, a pesar del hecho que los objetos de diferentes clases derivadas de una clase base, puedan diferenciarse de una manera significativa uno del otro, podemos crear su lista asociada (List) puesto que los consideramos como los objetos de la clase base. Pero lo invertido no es correcto: resulta que los objetos de la clase base no son los objetos de la clase derivada de una forma automática.

Se puede utilizar la conversión explícita de tipos para la transformación de los punteros de la clase base en los punteros de la clase derivada. Pero hay que estar completamente seguro de la admisibilidad de esta transformación, porque en el caso contrario surgirá un error crítico de tiempo de ejecución y el programa-mql5 se detendrá.

Véase también

Tipos de datos

Tipo void y constante NULL

Sintácticamente el tipo void es un tipo fundamental igual que el tipo char, uchar, bool, short, ushort, int, uint, color, long, ulong, datetime, float, double y string. Este tipo se utiliza o para indicar que la función no devuelve los valores, o como parámetro de la función que significa la falta de los parámetros.

La variable constante predeterminada **NULL** tiene el tipo *void*. Ésta puede ser asignada sin transformación a las variables de cualquier otro tipo fundamental. También se permite la comparación de las variables de tipos fundamentales con el valor **NULL**

Ejemplo:

```
//--- si la cadena no está inicializada, entonces le asignaremos nuestro valor predet
if(some_string==NULL) some_string="empty";
```

Además, **NULL** se puede comparar con los punteros a los objetos creados con el operador new.

Véase también

Variables, Funciones

Los punteros a objetos

En MQL5 existe la posibilidad de crear los objetos de tipo compuesto de una forma dinámica. Esto se hace con la ayuda del operador new, que devuelve el descriptor del objeto creado. El descriptor tiene el tamaño de 8 bytes. Sintácticamente los descriptores de objetos en MQL5 son parecidos a los punteros en C++.

Ejemplos:

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

Repitamos otra vez que a diferencia de C++ la variable *hobject* del ejemplo de arriba **no es el puntero a la memoria**, sino el descriptor de objeto.

Véase también

Variables, Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Referencias. Modificador & y palabra clave this

Traspaso de parámetros por referencia

En MQL5 los parámetros de tipos simples pueden ser traspasados por valor y por referencia, mientras que los parámetros de tipos compuestos siempre se traspasan por referencia. Para indicar al compilador la necesidad de traspaso de parámetros por referencia antes del nombre del parámetro se coloca el signo **&**.

El traspaso de parámetro por referencia significa el paso de dirección de una variable, por eso todas las modificaciones realizadas con el parámetro pasado por referencia en seguida se reflejarán también en la variable original. Al usar el traspaso de parámetros por referencia, se puede organizar al mismo tiempo el retorno de varios resultados desde la función. Para evitar los cambios del parámetro traspasado por referencia, hace falta utilizar el modificador const.

De esa manera, si el parámetro entrante de una función es un array, objeto de estructura o clase, entonces en el encabezado de la función después del tipo de variable y antes de su nombre se pone el signo '&'.

Ejemplo

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

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

En el ejemplo de arriba se declara la clase CDemoClass que contiene un elemento-array privado m_array[] del tipo double. Se declara la función setArray() en la que el array array[] es traspasado por referencia. Si escribimos el encabezado de la función sin indicar el traspaso por referencia, es decir, quitando el signo de ampersand, durante el intento de compilación de tal código saldrá un mensaje de error.

A pesar de que un array se traspasa por referencia, no podemos realizar la asignación de un array a otro. Es necesario hacer una copia elemento por elemento del contenido de array-fuente al array-receptor. Para los arrays y estructuras a la hora de traspaso en calidad del parámetro de la función la presencia del signo & es obligatoria durante la descripción de la función.

Palabra clave this

La variable del tipo clase (objeto) puede ser traspasada tanto por referencia, como por puntero. Un puntero, igual como una referencia, sirve para obtener el acceso a un objeto. Después de declarar el puntero a objeto es necesario aplicarle el operador new para su creación e inicialización.

La palabra reservada **this** sirve para obtener referencia del objeto a si mismo, dicha referencia debe ser accesible dentro de los métodos de la clase o estructura. **this** siempre hace referencia al objeto en el método del que se utiliza. Y la expresión GetPointer(this) da un puntero al objeto al que pertenece la función en la que se realiza la llamada a la función GetPointer(). En MQL5 las funciones no pueden devolver los objetos pero está permitido devolver el puntero a objetos.

Así, si hace falta que la función devuelva el objeto, podemos devolver el puntero a este objeto a modo de GetPointer(this). Añadamos a la descripción de la clase CDemoClass la función getDemoClass() que devuelve el puntero a objeto de esta clase.

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

public:
   void              setArray(double &array[]);
   CDemoClass       *getDemoClass();
  };
//+------------------------------------------------------------------+
//| relleno del array                                                |
//+------------------------------------------------------------------+
void  CDemoClass::setArray(double &array[])
  {
   if(ArraySize(array)>0)
     {
      ArrayResize(m_array,ArraySize(array));
      ArrayCopy(m_array,array);
     }
  }
//+------------------------------------------------------------------+
//| devuelve su propio puntero                                       |
//+------------------------------------------------------------------+
CDemoClass *CDemoClass::getDemoClass(void)
  {
   return(GetPointer(this));
  }
```

Las estructuras no disponen de punteros, no se les puede aplicar los operadores *new* y *delete*, tampoco se puede usar GetPointer(this.

Véase también

Punteros a objetos, Creación y eliminación de objetos, Visibilidad y tiempo de vida de variables

Operaciones y expresiones

Algunos caracteres y sucesiones de caracteres adquieren importancia especial. Son llamados símbolos de operaciones, por ejemplo:

```
+ - * / %     símbolos de operaciones aritméticas
&& ||         símbolos de operaciones lógicas
= += *=       símbolos de operaciones de asignación
```

Los símbolos de operaciones se utilizan en las expresiones y tienen sentido cuando se les dan los operandos correspondientes. Además, se da importancia especial a los signos de puntuación. Los signos de puntuación incluyen los paréntesis, llaves, coma, dos puntos y el punto coma.

Los símbolos de operaciones, signos de puntuación y espacios sirven para separar los elementos del lenguaje.

En este apartado se examinan los temas siguientes:

- Expresiones
- Operaciones aritméticas
- Operaciones de asignación
- Operaciones de relación
- Operaciones lógicas
- Operaciones a nivel de bits
- Otras operaciones
- Prioridades y orden de operaciones

Expresiones

Una expresión se compone de uno o varios operandos y símbolos de operaciones. Se puede escribirla en varias líneas.

Ejemplos:

```
a++; b = 10;          // varias expresiones se ubican en una línea
//---  una expresión se divide en varias líneas
x = (y * z) /
    (w + 2) + 127;
```

La expresión terminada con el punto y coma (;) es un operador.

Véase también

Prioridades y orden de las operaciones

Operaciones aritméticas

Las operaciones aditivas y multiplicativas pertenecen a las operaciones aritméticas:

```
Suma de valores                          i = j + 2;
Resta de valores                         i = j - 3;
Cambio de signo                          x = - x;
Multiplicación de valores                z = 3 * x;
Cociente de la división                  i = j / 5;
Resto de la división                     minutes = time % 60;
Suma de 1 al valor de la variable        i++;
Suma de 1 al valor de la variable        ++i;
Resta de 1 del valor de la variable      k--;
Resta de 1 del valor de la variable      --k;
```

Operaciones de incremento y decremento se aplican unicamente a las variables; no se usan con las constantes. El incremento prefijo (++i) y decremento prefijo (--k) se aplican a la variable justo antes de usarla en la expresión.

Post-incremento (i++) y post-decremento (k--) se aplican a la variable justo después de usarla en la expresión.

Observación importante

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

Pueden surgir problemas de cómputo a la hora de pasar la expresión arriba mencionada de un entorno de programación a otro (por ejemplo, de Borland C++ a MQL5). Generalmente el orden de cálculos depende de la implementación del compilador. En la práctica existen dos modos de implementar un post-decremento (post-incremento):

1. el post-decremento (post-incremento) se aplica a la variable tras calcular toda la expresión;
2. el post-decremento (post-incremento) se aplica a la variable directamente durante la operación.

En MQL5 en estos momentos está implementado el primer modo de calcular el post-decremento (post-incremento). Pero incluso teniendo este conocimiento, es mejor no experimentar con el uso de este detalle.

Ejemplos:

```
int a=3;
a++;             // expresión correcta
int b=(a++)*3;   // expresión incorrecta
```

Véase también

Prioridades y orden de las operaciones

Operaciones de asignación

El valor de la expresión que incluye la operación de asignación, es el valor del operando izquierdo después de la asignación.

```
Asignación del valor x a la variable y                                    y = x;
```

Las siguientes operaciones unen las operaciones aritméticas y a nivel de bits con la operación de asignación:

```
Sumando x al valor de la variable y                                       y += x;
Restando x del valor de la variable y                                     y -= x;
Multiplicando el valor de la variable y por x                             y *= x;
Dividiendo el valor de la variable y por x                                y /= x;
Resto de la división del valor de la variable y por x                     y %= x;
Desplaza x bits a la derecha la representación binaria de y               y >>= x;
Desplaza x bits a la izquierda la representacion binaria de y             y <<= x;
Operación a nivel de bits AND de las representaciones binarias y y x      y &= x;
Operación a nivel de bits OR de las representaciones binarias y y x       y |= x;
Operación a nivel de bits excluyendo OR
          de las representaciones binarias y y x                          y ^= x;
```

Operaciones a nivel de bits se realizan unicamente con números enteros. Durante la operación del desplazamiento lógico de la representación y a la derecha/izquierda sobre x bits, se usan 5 menores dígitos binarios del valor x, los dígitos mayores se omiten; es decir, el desplazamiento se realiza sobre 0-31 bits.

Durante la operación %= (valor y por el módulo x) el signo del resultado coincide con el signo del dividiendo.

El operador de asignación puede aparecer varias veces en una expresión. En este caso el procesamiento de la expresión se realiza de derecha a izquierda:

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

Al principio a la variable x se le asigna el valor 3, luego a la variable y se le va a asignar el valor de la variable x, es decir, también 3.

Véase también

Prioridades y orden de las operaciones

Operaciones de relación

El valor lógico FALSO se representa por el valor entero cero, y el valor VERDADERO se representa por cualquier valor distinto a cero.

FALSO (0) y VERDADERO (1) son valores de las expresiones que contienen las operaciones de relación o las operaciones lógicas.

```
Verdadero, si a es igual a b                    a == b;
Verdadero, si a no es igual a b                 a != b;
Verdadero, si a es menor que b                   a < b;
Verdadero, si a es mayor que b                   a > b;
Verdadero, si a es menor o igual a b          a <= b;
Verdadero, si a es mayor o igual a b          a >= b;
```

No se puede comparar la equivalencia de dos números reales. En mayoría de los casos resulta que dos números aparentemente iguales pueden ser desiguales por la diferencia del valor en el décimoquinto dígito después de la coma. Para una comparación correcta de dos números reales es necesario comparar la diferencia normalizada de estos números con el valor cero.

Ejemplo:

```
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  %.16f   son iguales",first,third);
     }
  }
// Resultado: 0.3000000000000000  0.2999999999999998   no obstante, son iguales
```

Véase también

Prioridades y orden de las operaciones

Operaciones lógicas

Negación lógica NOT(!)

El operando de la operación de negación lógica NOT(!) debe tener el tipo aritmético. El resultado es igual a VERDADERO(1) si el valor del operando es FALSO(0), y es igual a FALSO(0) si el operando no es igual a FALSO(0).

```
if(!a) Print("no 'a'");
```

Operación lógica OR (||)

Operación lógica OR (||) de los valores x y y. El valor de la expresión es VERDADERO(1) si el valor de x o y es verdadero (no cero). En caso contrario es FALSO(0).

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

Operación lógica AND (&&)

Operación lógica AND (&&) de los valores x y y. El valor de la expresión es VERDADERO(1) si el valor de x y y son verdaderos (no cero). En caso contrario es FALSO(0).

Breve estimación de operaciones lógicas

Tratándose de operaciones lógicas, se les aplica un esquema llamado "de breve estimación". Esto quiere decir que el cálculo de una operación lógica se finaliza cuando su resultado puede ser estimado de forma precisa.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- el primer ejemplo de breve estimación
   if(func_false() && func_true())
     {
      Print("Operación &&: Este mensaje nunca estará visible para Ud.");
     }
   else
     {
      Print("Operación &&: El resultado de la primera expresión es false, por eso la
     }
//--- el segundo ejemplo de breve estimación
   if(!func_false() || !func_true())
     {
      Print("Operación ||: El resultado de la primera expresión es true, por eso la s
     }
   else
```

```
     {
      Print("Operación ||: Este mensaje nunca estará visible para Ud.");
     }
  }
//+------------------------------------------------------------------+
//| la función siempre devuelve false                                |
//+------------------------------------------------------------------+
bool func_false()
  {
   Print("Función func_false()");
   return(false);
  }
//+------------------------------------------------------------------+
//| la función siempre devuelve true                                 |
//+------------------------------------------------------------------+
bool func_true()
  {
   Print("Función func_true()");
   return(true);
  }
```

Véase también

Prioridades y orden de las operaciones

Operaciones a nivel de bits

Complemento a uno

Complemento del valor de la variable a uno. El valor de la expresión contiene 1 en todos los dígitos donde el valor de la variable contiene 0, y 0 en todos los dígitos donde el valor de la variable contiene 1.

```
b = ~n;
```

Ejemplo:

```
   char a='a',b;
   b=~a;
   Print("a = ",a, "  b = ",b);
// Resultado va a ser el siguiente::
// a = 97   b = -98
```

Desplazamiento a la derecha

La representación binaria de x se desplaza a la derecha a y dígitos. Si el valor desplazado posee el tipo sin signos, entonces se realiza el desplazamiento lógico a la derecha, es decir, los dígitos liberados por la izquierda se rellenan con ceros.

Pero si el valor desplazado tiene el tipo de signo, entonces se realiza el desplazamiento aritmético a la derecha, es decir, los dígitos liberados por la izquierda van a ser rellenados con el valor del bit de signo (si el número es positivo, el valor del bit de signo es igual a 0; si el número es negativo, el valor del bit de signo es igual a 1)

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

Ejemplo:

```
   char a='a',b='b';
   Print("Before:  a = ",a, "  b = ",b);
//--- hagamos el desplazamiento a la derecha
   b=a >> 1;
   Print("After:   a = ",a, "  b = ",b);
// Resultado va a ser el siguiente:
// Before:  a = 97   b = 98
// After:   a = 97   b = 48
```

Desplazamiento a la izquierda

La representación binaria de x se desplaza a la izquierda a y dígitos, rellenándose los dígitos liberados por la derecha con ceros.

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

Ejemplo:

```
   char a='a',b='b';
   Print("Before:  a = ",a, "  b = ",b);
```

```
//--- hagamos el desplazamiento a la izquierda
   b=a << 1;
   Print("After:   a = ",a, "  b=",b);
// Resultado va a ser el siguiente:
// Before:  a = 97   b = 98
// After:   a = 97   b = -62
```

No se recomienda realizar el desplazamiento a más o igual cantidad de bits que la longitud de la variable desplazada, puesto que el resultado de dicha operación no está definido.

Operación a nivel de bits AND

Operación a nivel de bits AND de las representaciones binarias y y x. El valor de la expresión contiene 1 (VERDADERO) en todos los dígitos en los que como x, tanto y no contienen ceros; y 0 (FALSO) en los demás dígitos.

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

Ejemplo:

```
   char a='a',b='b';
//--- operación AND
   char c=a&b;
   Print("a = ",a,"  b = ",b);
   Print("a & b = ",c);
// Resultado va a ser el siguiente:
// a = 97   b = 98
// a & b = 96
```

Operación a nivel de bits OR

Operación a nivel de bits OR de las representaciones binarias y y x. El valor de la expresión contiene 1 en todos los dígitos en los que x o y no contienen 0; y 0 en todos los demás dígitos.

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

Ejemplo:

```
   char a='a',b='b';
//--- operación OR
   char c=a|b;
   Print("a = ",a,"  b = ",b);
   Print("a | b = ",c);
// Resultado va a ser el siguiente:
// a = 97   b = 98
// a | b = 99
```

Operación a nivel de bits excluyendo OR

Operación a nivel de bit excluyendo AND (eXclusive OR) de las representaciones binarias y y x. El valor de la expresión contiene 1 en los dígitos en los que x y y poseen los valores binarios diferentes; y 0 en todos los demás dígitos.

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

Ejemplo:

```
   char a='a', b='b';
//--- operación excluyendo OR
   char c=a^b;
   Print("a = ",a,"  b = ",b);
   Print("a ^ b = ",c);
// Resultado va a ser el siguiente:
// a = 97   b = 98
// a ^ b = 99
```

Operaciones a nivel de bits se realizan unicamente con los números enteros.

Véase también

Prioridades y orden de las operaciones

Otras operaciones

Indexación ([])

A la hora de dirigirse al elemento número i del array, el valor de la expresión será el valor de la variable con el número i.

Ejemplo:

```
array[i] = 3; // Adjudicar el valor 3 al elemento número i del array array.
```

Sólo el número entero puede ser un índice del array. Se admiten sólo los arrays cuatro-dimensionales. La indexación de cada medición se realiza desde 0 hasta **el tamaño de medición**-1. En un caso particular de un array unidimensional compuesto por 50 elementos, la referencia hacia el primer elemento se verá como el array[0], hacia el último elemento - array[49].

Al acceder fuera de los límites del array, el subsistema ejecutivo generará un error crítico y la ejecución del programa se detendrá.

Llamada a función con los argumentos x1, x2,..., xn

Cada argumento puede representar una constante, variable o expresión del tipo correspondiente. Los argumentos traspasados se separan con comas y deben encerrarse entre los paréntesis, el paréntesis que abre debe seguir detrás del nombre de la función que se llama.

El valor de la función es el valor que se devuelve a la función. Si el tipo del valor devuelto a la función es void, no se puede colocar la llamada a esta función en la parte derecha dentro de la operación de asignación. ¡Atención!, las expresiones x1,..., xn serán ejecutadas precisamente en este orden.

Ejemplo:

```
   int length=1000000;
   string a="a",b="b",c;
//---
   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);
```

Operación coma (,)

Las expresiones separadas con coma se calculan de izquierda a derecha. Todos los efectos secundarios derivados del cálculo de la expresión izquierda pueden surgir antes de calcular la expresión de la

derecha. El tipo y el valor del resultado coinciden con el tipo y el valor de la expresión derecha. Como ejemplo podemos estudiar la lista de los parámetros traspasados (véase más arriba).

Ejemplo:

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

Operación punto (.)

Para el acceso directo a los elementos públicos de las estructuras y clases se utiliza la operación punto. Sintaxis

```
Nombre_de_la_variable_tipo_estructura.Nombre_del_elemento
```

Ejemplo:

```
   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;
```

Operación de resolución de contexto (::)

En el programa mql5 cada función tiene su contexto de ejecución. Por ejemplo, la función del sistema Print() se ejecuta en el contexto global. Las funciones importadas se llaman en el contexto de importación correspondiente. Las funciones-métodos de las clases tienen el contexto de la clase correspondiente. La sintaxis de la operación de resolución de contexto:

```
[Nombre_del_contexto]::Nombre_de_función(parámetros)
```

Si falta el nombre del contexto, es una indicación directa a la utilización del contexto global. En caso de ausencia de la operación de resolución de contexto la función se busca en el contexto más próximo. Si en el contexto local no hay ninguna función, la busqueda se realiza en el contexto global.

Además, la operación de resolución de contexto se usa para definir la función-miembro de la clase.

```
tipo Nombre_de_clase::Nombre_de_función(descripción_de_parámetros)
   {
// cuerpo de la función
   }
```

Ejemplo:

```
#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();
  }
//+------------------------------------------------------------------+
```

Operación de obtención del tamaño de tipos de datos o el tamaño de objeto de cualquier tipo de datos (sizeof)

Utilizando la operación sizeof se puede determinar el tamaño de la memoria que corresponde al identificador o al tipo. La operación sizeof tiene el formato siguiente:

Ejemplo:

```
sizeof(expresión)
```

Cualquier identificador o el nombre del tipo encerrado entre paréntesis puede ser utilizado como una expresión. Cabe mencionar que no se puede usar el nombre del tipo void, y el identificador no puede pertenecer al campo de bits, o ser el nombre de la función.

Si la expresión es el nombre del array estático (es decir, se da la primera dimensión), entonces el resultado es el tamaño de todo el array (es decir, producto de multiplicación del número de elementos por longitud del tipo). Si la expresión es el nombre del array dinámico (no se da la primera dimensión), entonces el resultado es el tamaño del objeto del array dinámico.

Cuando sizeof se aplica al nombre del tipo de la estructura o clase, o al identificador que posee el tipo de estructura o clase, el resultado es el tamaño real de la estructura o clase.

Ejemplo:

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

El cálculo del tamaño se realiza durante la fase de compilación.

Véase también

Prioridades y orden de las operaciones

Prioridades y orden de las operaciones

Para cada grupo de operaciones de la tabla la prioridad es la misma. Cuanto más alta sea la prioridad del grupo, más arriba se encuentra éste en la tabla. El orden de ejecución determina la agrupación de operaciones y operandos.

Atención: La prioridad de ejecución de las operaciones en el lenguaje MQL5 corresponde a la prioridad adoptada en C++ y se diferencia de la prioridad aceptada en el lenguaje MQL4.

Operación	Descripción	Orden de ejecución
() [] .	Llamada a la función Selección del elemento del array Selección del elemento de la estructura	De izquierda a derecha
! ~ - ++ -- (tipo) sizeof	Negación lógica Negación a nivel de bits (complement) Cambio de signo Incremento a uno (increment) Decremento a uno (decrement) Conversión de tipo Cálculo del tamaño en bytes	De derecha a izquierda
* / %	Multiplicación División División inexacta	De izquierda a derecha
+ -	Suma Resta	De izquierda a derecha
<< >>	Desplazamiento a la izquierda Desplazamiento a la derecha	De izquierda a derecha
< <= > >=	Menos que Menos o igual Más que Más o igual	De izquierda a derecha
== !=	Igual a No es igual a	De izquierda a derecha
&	Operación a nivel de bits AND	De izquierda a derecha
^	Operación a nivel de bits excluyendo OR (eXclude OR)	De izquierda a derecha
|	Operación a nivel de bits OR	De izquierda a derecha
&&	Operación lógica AND	De izquierda a derecha
||	Operación lógica OR	De izquierda a derecha

?:	Operación condicional	De derecha a izquierda
= *= /= %= += -= <<= >>= &= ^= |=	Asignación Multiplicación con asignación División con asignación División inexacta con asignación Suma con asignación Resta con asignación Desplazamiento a la izquierda con asignación Desplazamiento a la derecha con asignación A nivel de bits AND con asignación Excluyendo OR con asignación A nivel de bits OR con asignación	De derecha a izquierda
,	Coma	De izquierda a derecha

Para cambiar el orden de ejecución de la operación se utilizan los paréntesis que tengan la más alta prioridad.

Operadores

Los operadores del lenguaje describen algunas acciones algorítmicas que tienen que ser ejecutadas para resolver el problema. El cuerpo del programa es una secuencia de estos operadores. Los operadores siguen uno detrás del otro y se separan con el punto y coma.

Operador	Descripción
Operador compuesto {}	Uno o más operadores de cualquier tipo encerrados entre las llaves { }
Operador-expresión (;)	Cualquier expresión que se termina con el punto y coma (;)
Operador return	Termina la ejecución de la función corriente y devuelve el control al programa iniciador
Operador condicional if-else	Se usa si hay necesidad de hacer una elección
Operador condicional ?:	Un análogo más simple del operador condicional if-else
Operador de selección switch	Pasa el control al operador que corresponde al valor de la expresión
Operador cíclico while	Ejecuta un operador hasta que la expresión verificada no sea falsa. El chequeo de la expresión se realiza antes de cada iteración
Operador cíclico for	Ejecuta un operador hasta que la expresión verificada no sea falsa. El chequeo de la expresión se realiza antes de cada iteración
Operador cíclico do-while	Ejecuta un operador hasta que la expresión verificada no sea falsa. El chequeo de la condición del fin se realiza al final, después de cada ronda del ciclo. El cuerpo del ciclo se ejecuta por lo menos una vez.
Operador break	Suspende la ejecución del operador externo adjunto más próximo switch, while, do-while o for
Operador continue	Pasa el control al inicio del operador cíclico externo más próximo while, do-while o for
Operador new	Crea el objeto del tamaño correspondiente y devuelve el descriptor del objeto creado.
Operador delete	Elimina el objeto creado por el operador new.

Un operador puede ocupar una o más líneas. Dos o más operadores pueden ubicarse en una línea. Los operadores que controlan el orden de ejecución (if, if-else, switch, while y for) pueden ser insertados uno en otro.

Ejemplo:

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

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador compuesto

Operador compuesto (bloque) se compone de uno o más operadores de cualquier tipo encerrados entre las llaves { }. Después de la llave que cierra no puede haber el punto y coma (;).

Ejemplo:

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

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador-expresión

Cualquier expresión que se termina con punto y coma (;) es un operador. Más abajo vienen los ejemplos de los operadores-expresiones.

Operador de asignación:

Identificador = expresión;

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

En una expresión el operador de asignación puede usarse varias veces. En este caso la expresión se procesa de derecha a izquierda:

Operador de llamada a función:

Nombre_de_función (argumento1,..., argumentoN);

```
FileClose(file);
```

Operador vacío

Se compone únicamente del punto y coma (;) y se usa para determinar el cuerpo vacío del operador de control.

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador de retorno return

El operador return termina la ejecución de la función corriente y devuelve el control al programa iniciador. El resultado del cálculo de expresión es retornado a la función invocada. La expresión puede contener un operador de asignación.

Ejemplo:

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

En las funciones con tipo de valor devuelto void es necesario usar el operador return sin expresión:

```
void SomeFunction()
  {
   Print("Hello!");
   return;     // se puede eliminar este operador
  }
```

La llave derecha de la función comprende la ejecución implícita del operador return sin expresión.

Se puede devolver los tipos simples, estructuras simples, punteros a objetos. Mediante el operador *return* no se puede devolver cualquier array, objetos de clases, variables del tipo de estructuras compuestas.

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador condicional if-else

El operador IF - ELSE se usa cuando surge la necesidad de hacer una elección. Formalmente, la sintaxis es así:

```
if (expresión)
     operador1
else
     operador2
```

Si la expresión es verdadera, se ejecuta el operador1 y el control se pasa al operador que sigue después del operador2 (es decir, el operador2 no se ejecuta). Si la expresión es falsa, se ejecuta el operador2.

La parte else del operador if se puede omitir. Por eso puede surgir una ambigüedad en los operadores interiores if con la parte omitida else. En este caso, else se vincula con anterior operador if más cercano en el mismo bloque, y que no tiene la parte else.

Ejemplos:

```
//--- La parte else se refiere al segundo operador if:
if(x>1)
   if(y==2) z=5;
else     z=6;
//--- La parte else se refiere al primer operador if
if(x>1)
  {
   if(y==2) z=5;
  }
else        z=6;
//--- Operador interior
if(x=='a')
  {
   y=1;
  }
else if(x=='b')
  {
   y=2;
   z=3;
  }
else if(x=='c')
  {
   y=4;
  }
else Print("ERROR");
```

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador condicional ?:

La forma general de un operador ternario es la siguiente:

```
expresión1? expresión2:expresión3
```

Como el primer operando - "expresión1" se puede utilizar cualquier expresión, el resultado de la cual es el valor del tipo bool. Si el resultado es true, entonces se ejecuta el operador asignado por el segundo operando, es decir, "expresión2".

Si el primer operando es igual a false, entonces se ejecuta el tercer operando - "expresión3". El segundo y el tercer operando, es decir, "expresión2" y "expresión3" tienen que retornar los valores del mismo tipo y no deben tener el tipo void. El resultado de ejecución del operador condicional es el resultado de la expresión2 o el de la expresión3, dependiendo del resultado de la expresión1.

```
//--- normalizamos la diferencia entre los precios de apertura y de cierre para el ra
double true_range = (High==Low)?0:(Close-Open)/(High-Low);
```

Esta entrada es equivalente a la siguiente

```
   double true_range;
   if(High==Low)true_range=0;                // si High y Low son iguales
   else true_range=(Close-Open)/(High-Low); // si el rango no es cero
```

Limitaciones en el uso del operador

A base del valor "expresión1" el operador tiene que devolver uno de dos valores - "expresión2" o "expresión3". Existe una serie de limitaciones para estas expresiones:

1. No se puede confundir el tipo definido por el usuario con el tipo simple o enumeración. Para el puntero se puede utilizar NULL.
2. Si los tipos de los valores son simples, entonces el tipo del operador será el tipo máximo (véase Conversión de tipos).
3. Si uno de los valores tiene el tipo de enumeración y el segundo es del tipo numérico, la enumeración se reemplace con int y se aplica la segunda regla.
4. Si los dos valores son valores de enumeración, entonces sus tipos tienen que ser iguales, y el tipo del operador será la enumeración.

Limitaciones para los tipos definidos por el usuario (clases o estructuras):

a) los tipos tienen que ser idénticos o uno debe heredarse del otro.

b) si los tipos no son idénticos (la herencia), entonces el hijo se convierte al padre de forma implícita, es decir el tipo del operador va a ser del tipo del padre.

c) no se puede confundir el objeto y el puntero - o las dos expresiones son objetos, o bien los punteros. Se puede usar NULL para el puntero.

Nota

Tenga cuidado a la hora de usar el operador condicional como un argumento de la función sobrecargada, porque el tipo del resultado del operador condicional se determina en el momento de

compilación del programa. Y este tipo se determina como el más grande de los tipos "expresión2" y "expresión3".

Ejemplo:

```
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);   // recibimos un aviso del compilador
   func(!Expression1?Expression2:Expression3);  // recibimos un aviso del compilador
  }

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

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador switch

Compara el valor de expresión con las constantes en todas las variantes de *case* y pasa el control al operador correspondiente al valor de la expresión. Cada variante *case* puede ser marcada con la constante entera, constante de signo o expresión constante. La expresión constante no puede contener las variables o llamadas a funciones. La expresión del operador *switch* tiene que ser del tipo entero.

```
switch(expresión)
  {
   case constante: operadores
   case constante: operadores
      ...
   default: operadores
  }
```

Los operadores vinculados a la marca *default* se ejecutan si ninguna de las constantes en los operadores *case* no es igual al valor de la expresión. La variante *default* no tiene que ser declarada obligatoriamente y no tiene que ser última obligatoriamente. Si ninguna de las constantes corresponde al valor de la expresión y la variante *default* no está presente, no se ejecuta ninguna acción.

La palabra clave *case* y la constante son sólo marcas, y si los operadores se ejecutan para una variante *case*, entonces más adelante van a ejecutarse los operadores de todas las variantes posteriores hasta que se llegue al operador *break*. Eso permite vincular una sucesión de operadores con varias variantes.

Una expresión constante se calcula durante el proceso de compilación. Nunca dos constantes en el mismo operador *switch* pueden tener los mismos valores.

Ejemplos:

```
//--- el primer ejemplo
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;
  }

//---  el segundo ejemplo
   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);
/*
   Resultado
   default
*/
```

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador cíclico while

El operador while se compone de la expresión chequeada y el operador que tiene que ser ejecutado:

```
while(expresión)
  operador;
```

Si la expresión es verdadera, entonces el operador se ejecuta hasta que la expresión se haga falsa. Si la expresión es falsa, entonces el control se pasa al siguiente operador. El valor de la expresión se define antes de que el operador sea ejecutado. Por tanto, si la expresión es falsa desde el principio, entonces el operador no será ejecutado en absoluto.

Nota

Si se supone procesar una gran cantidad de iteraciones en el ciclo, entonces se recomienda comprobar el hecho de finalización forzosa del programa por medio de la función IsStopped().

Ejemplo:

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

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador cíclico for

El operador for se compone de tres expresiones y un operador ejecutable:

```
for(expresión1; expresión2; expresión3)
   operador;
```

Expresión1 describe la inicialización del ciclo. Expresión2 comprueba la condición del fin del ciclo. Si es verdadera, el operador del cuerpo de ciclo for se ejecuta. Todo se repite hasta que la expresión2 se haga falsa. Si es falsa, el ciclo se termina y el control se pasa al siguiente operador. Expresión3 se calcula después de cada iteración.

El operador for equivale a la siguiente sucesión de operadores:

```
expresión1;
while(expresión2)
  {
   operador;
   expresión3;
  };
```

En operador for puede faltar cualquiera de las tres expresiones o todas, sin embargo, no se puede omitir los puntos y comas (;) que las separan. Si la expresión2 se omite, se considera que es siempre verdadera. El operador for(;;) es un ciclo infinito equivalente al operador while(1). Cada expresión1 y 3 puede componerse de varias expresiones unidas por el operador coma ','.

Nota

Si se supone procesar una gran cantidad de iteraciones en el ciclo, entonces se recomienda comprobar el hecho de finalización forzosa del programa por medio de la función IsStopped().

Ejemplos:

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

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de

objetos

Operador cíclico do while

Los ciclos for y while realizan la comprobación de terminación al principio del ciclo y no en su final. El tercer operador del ciclo do - while comprueba la condición de terminación al final, después de cada iteración del ciclo. El cuerpo del ciclo siempre se ejecuta por lo menos una vez.

```
do
   operador;
while(expresión)
```

Al principio se ejecuta el operador, luego se calcula la expresión. Si la expresión es verdadera, entonces el operador vuelve a ejecutarse, etc. Si la expresión se hace falsa, el ciclo se acaba.

Nota

Si se supone procesar una gran cantidad de iteraciones en el ciclo, entonces se recomienda comprobar el hecho de finalización forzosa del programa por medio de la función IsStopped().

Ejemplo:

```
//--- el cálculo de la sucesión de Fibonacci
   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++; // Sin este operador se obtiene el ciclo infinito!
     }
   while(i<counterFibonacci && !IsStopped());
```

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador break

El operador break termina la ejecución del más cercano operador adjunto externo switch, while, do-while o for. El control se pasa al operador que sigue después del terminado. Uno de los objetivos de este operador consiste en terminar el ciclo al adjudicar un valor concreto a una variable.

Ejemplo:

```
//--- la búsqueda del primer elemento cero
for(i=0;i<array_size;i++)
  if(array[i]==0)
    break;
```

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador de continuación continue

El operador continue pasa el control al inicio del operador externo de ciclo más cercano while, do-while o for, provocando así el comienzo de la siguiente iteración. Por su acción, este operador es contrario al operador break.

Ejemplo:

```
//--- suma de todos los elementos no nulos
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);
  }
```

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador-creador de objetos new

El operador new crea automáticamente un objeto de tamaño correspondiente, arranca el constructor del objeto y devuelve el descriptor del objeto creado. En caso de un fallo el operador devuelve el descriptor cero que se puede comparar con la constante NULL.

El operador *new* puede ser aplicado únicamente a los objetos de la clase, no se puede aplicarlo a las estructuras.

El operador no se aplica para crear los arrays de objetos. Para eso hay que usar la función ArrayResize ().

Ejemplo:

```
//+------------------------------------------------------------------+
//| Creación de una figura                                           |
//+------------------------------------------------------------------+
void CTetrisField::NewShape()
  {
   m_ypos=HORZ_BORDER;
//--- de una manera aleatoria creamos una de 7 posibles figuras
   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;
     }
//--- dibujamos
   if(m_shape!=NULL)
     {
      //--- configuraciones iniciales
      m_shape.SetRightBorder(WIDTH_IN_PIXELS+VERT_BORDER);
      m_shape.SetYPos(m_ypos);
      m_shape.SetXPos(VERT_BORDER+SHAPE_SIZE*8);
      //--- dibujamos
      m_shape.Draw();
     }
//---
  }
```

Cabe mencionar que el descriptor del objeto no es un puntero a la memoria.

El objeto creado con operador new tiene que ser eliminado de una forma explícita por el operador delete.

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Operador-eliminador de objetos delete

El operador delete elimina un objeto creado por el operador new, arranca el destructor de clase correspondiente y libera la memoria ocupada por el objeto. El descriptor real de un objeto existente se utiliza como un operando. Una vez terminada la operación delete el descriptor del objeto pierde su validez.

Ejemplo:

```
   //--- eliminamos la figura colocada
   delete m_shape;
   m_shape=NULL;
   //--- creamos una figura nueva
   NewShape();
```

Véase también

Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Funciones

Cualquier tarea puede ser dividida en unas subtareas, cada una de las cuales bien puede ser representada directamente en forma de un código o bien dividida en subtareas más pequeñas. Este método se llama *refinamiento sucesivo*. Las funciones sirven para escribir el código de programación de estas tareas a resolver. El código que describe lo que hace la función se llama *definición de función*:

```
encabezado_de_función
  {
   instrucción
  }
```

Todo lo que se encuentra antes de la primera llave es el encabezado de la definición de función, lo que viene entre las llaves es *el cuerpo* de la definición de función. El encabezado de la función contiene la descripción del tipo de valor devuelto, nombre del (identificador) y parámetros formales. La cantidad de parámetros traspasados a una función está limitada y no puede superar 64.

Se puede llamar a la función de otras partes del programa tantas veces como haga falta. En realidad, el tipo devuelto, identificador de la función y tipos de parámetros constituyen *prototipo de función*.

Un prototipo de función es la declaración de una función pero no su definición. Gracias a la declaración explícita del tipo devuelto y de la lista de tipos de argumentos, la prueba rigurosa de tipos y las conversiones implícitas de tipos son posibles durante la invocación de las funciones. Sobre todo, las declaraciones de funciones se utilizan muy a menudo en las clases para mejorar la lectura del código fuente.

La definición de función tiene que corresponder exactamente a su declaración. Cada función declarada tiene que estar definida.

Ejemplo:

```
double                          // tipo del valor devuelto
linfunc (double a, double b) // nombre de función y lista de parámetros
  {
                                // operador compuesto
   return (a + b);              // valor devuelto
  }
```

El operador return puede devolver el valor de la expresión que se encuentra en este operador. Si hace falta, el valor de la expresión se convierte al tipo del resultado de función. La función que no devuelve el valor tiene que ser descrita como la que tiene el tipo void.

Ejemplo:

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

Los parámetros traspasados a la función pueden tener los valores por defecto que se definen por las constantes del tipo correspondiente.

Ejemplo:

```
int somefunc(double a,
             double d=0.0001,
             int n=5,
             bool b=true,
             string s="passed string")
  {
   Print("Parámetro obligatorio a=",a);
   Print("Se han pasado los siguientes parámetros: d = ",d," n = ",n," b = ",b," s =
   return(0);
  }
```

Si algún parámetro ha obtenido el valor default, todos los parámetros siguientes también deben tener el valor default.

Ejemplo de una declaración errónea:

```
int somefunc(double a,
             double d=0.0001,    // se ha declarado el valor por defecto 0.0001
             int n,              // el valor por defecto no aparece !
             bool b,             // el valor por defecto no aparece !
             string s="passed string")
  {
  }
```

Véase también

Sobrecarga, Funciones virtuales, Polimorfismo

Llamada a una función

Si un nombre que no ha sido descrito antes aparece en la expresión y le sigue el paréntesis izquierdo, entonces según el contexto éste se considera el nombre de una función.

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

Los argumentos (parámetros formales) se traspasan por valor, es decir, cada expresión x1, . . . , xn se calcula, y su valor es pasado a la función. El orden del cálculo de expresión y el orden de la carga de valores no se garantizan. Durante la ejecución se realiza el chequeo del número y el tipo de argumentos pasados a la función. Este modo de dirigirse a una función se llama la llamada por valor.

La llamada a una función es una expresión cuyo valor es el valor devuelto por la función. El tipo de función descrito tiene que corresponder al tipo del valor devuelto. La función puede ser declarada o descrita en cualquier parte del programa a nivel global, es decir, fuera de otras funciones. Una función no puede ser declarada o descrita dentro de otra función.

Ejemplos:

```
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);
  }
```

Al invocar una función que tiene los parámetros predefinidos, la lista de parámetros pasados puede ser limitada, pero no antes del primer parámetro predefinido.

Ejemplos:

```
void somefunc(double init,
              double sec=0.0001, //los valores por defecto están definidos
              int level=10);
//...
somefunc();                      // llamada errónea. tiene que haber el primer parame
somefunc(3.14);                  // llamada correcta
somefunc(3.14,0.0002);           // llamada correcta
somefunc(3.14,0.0002,10);        // llamada correcta
```

Al llamar a una función, no se puede omitir los parámetros, aunque éstos tengan los valores predefinidos:

```
somefunc(3.14, , 10);           // llamada errónea -> el segundo parámetro está omiti
```

Véase también

Sobrecarga, Funciones virtuales, Polimorfismo

Traspaso de parámetros

Existen dos métodos mediante los cuales el lenguaje de programación puede pasar el argumento al subprograma (función). El primer modo consiste en traspasar los parámetros por valor. Este método copia el valor del argumento en el parámetro formal de la función. Por tanto, cualquieras que sean las modificaciones de este parámetro dentro de la función, no tendrán ninguna influencia sobre el correspondiente argumento de llamada.

```
//+------------------------------------------------------------------+
//| traspaso de parámetros por valor                                 |
//+------------------------------------------------------------------+
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 y b antes de la llamada:",a," ",b);
   double d=FirstMethod(a,b);
   Print("a y b después de la llamada:",a," ",b);
  }
//--- resultado de ejecución del script
//  a y b antes de la llamada: 14 8
//  a y b después de la llamada: 14 8
```

El segundo modo es el traspaso de parámetros por referencia. En este caso la referencia al parámetro (y no su valor) se pasa al parámetro de la función. Se usa dentro de la función para dirigirse al parámetro actual indicado en la llamada. Esto significa que las modificaciones del parámetro van a influir sobre el argumento utilizado para invocar la función.

```
//+------------------------------------------------------------------+
//| traspaso de parámetros por referencia                            |
//+------------------------------------------------------------------+
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 y b antes de la llamada:",a," ",b);
   double d=SecondMethod(a,b);
   Print("a y b después de la llamada:",a," ",b);
  }
//+------------------------------------------------------------------+
//--- resultado de ejecución del script
//  a y b antes de la llamada: 14 8
//  a y b después de la llamada: 28 4
```

MQL5 utiliza los dos métodos, salvo una excepción: los arrays, variables del tipo de estructuras y objetos de clases siempre se traspasan por referencia. Para excluir los cambios de los parámetros actuales (argumentos pasados al invocar una función) es necesario usar el especificador de acceso const. Cuando se intenta cambiar el contenido de la variable que ha sido declarada con el especificador *const*, el compilador mostrará un error.

Nota

Hay que recordar que los parámetros se traspasan a una función del revés, es decir, al principio se calcula y se pasa el último parámetro, luego el penúltimo, etc. En último lugar se calcula y se pasa el parámetro que está primero después de las llaves.

Ejemplo:

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

   func(a[i],a[i++],"La primera llamada(i = "+string(i)+")");
   func(a[i++],a[i],"La segunda llamada(i = "+string(i)+")");
// Resultado:
// Primera llamada(i=0) : par1 = 1     par2 = 0
// Segunda llamada(i=1) : par1 = 1     par2 = 1

  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
```

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

Durante la primera llamada en el ejemplo mencionado, al principio la variable ***i*** toma parte en la concatenación de cadenas de caracteres

```
   "La primera llamada(i = "+string(i)+")"
```

además allí su valor no se cambia. Luego la variable **i** toma parte en el calculo del elemento del array ***a[i++]***, es decir, después de tomar el elemento i del array, la variable ***i*** se incrementa. Y sólo después de eso se calcula el primer parámetro con el valor cambiado de la variable ***i***.

En la segunda llamada durante el cálculo de todos los tres parámetros, se usa el mismo valor de i que ha sido cambiado durante la primera invocación de la función, y sólo una vez calculado el primer parámetro, la variable ***i*** vuelve a cambiarse.

Véase también

Visibilidad y tiempo de vida de variables, Sobrecarga, Funciones virtuales, Polimorfismo

Sobrecarga de funciones

Por lo común, en el nombre de una función intentan reflejar su principal finalidad. Por regla general, los programas legibles contienen diversos identificadores bien seleccionados. A veces, diferentes funciones se usan con los mismos propósitos. Vamos a considerar, por ejemplo, la función que calcula el valor promedio del array de números de doble precisión, y la misma función pero la que opera con el array de números enteros. Es cómodo nombrar ambas funciones AverageFromArray:

```
//+------------------------------------------------------------------+
//| cálculo del promedio para el array de tipo double                |
//+------------------------------------------------------------------+
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];    // suma para double
     }
   aver=sum/size;       // simplemente dividimos la suma por la cantidad
//---
   Print("Cálculo de la media para un array del tipo double");
   return aver;
  }
//+------------------------------------------------------------------+
//| cálculo del promedio para el array de tipo int                   |
//+------------------------------------------------------------------+
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];     // suma para int
     }
   aver=(double)sum/size;// convertimos la suma al tipo double y la dividimos
//---
   Print("Cálculo de la media para un array del tipo int");
   return aver;
  }
```

Cada función contiene la extracción del mensaje a través de la función Print();

```
   Print("Cálculo de la media para un array del tipo int");
```

El compilador elige una función necesaria de acuerdo con los tipos de argumentos y su cantidad. La regla, según la cual se hace la elección, se llama *algoritmo de equivalencia a la signatura*. Una signatura es una lista de tipos usados en la declaración de función.

Ejemplo:

```
//+------------------------------------------------------------------+
//| 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);
  }
//--- Resultado de funcionamiento del script
// cálculo del promedio para el array de tipo int
// cálculo del promedio para el array de tipo double
// int_aver = 3.00000000    double_aver = 3.30000000
```

La sobrecarga es la práctica de asignación de varios valores a una función. La elección de un valor específico depende de los tipos de argumentos recibidos por la función. Una función específica se elige a base de la correspondencia de la lista de argumentos durante la invocación de la función a la lista de parámetros en la declaración de la función.

Cuando se invoca una función sobrecargada, el compilador debe tener un algoritmo para elegir una función apropiada. El algoritmo que hace la elección depende del tipo de conversiones que estén presentes. La mejor correlación tiene que ser única. Tiene que ser la mejor por lo menos para uno de los argumentos, e igual de buena como las demás correspondencias para todos los otros argumentos.

Más abajo viene el algoritmo de correspondencia para cada argumento.

Algoritmo de elección de una función sobrecargada

1. Usar la correspondencia estricta (si es posible).
2. Intentar el incremento estándar del tipo.
3. Intentar la transformación estándar del tipo.

El incremento estándar del tipo es mejor que las demás transformaciones estándar. El incremento es una transformación float en double, y también bool, char, short o enum en int. Además, a las transformaciones estándar pertenecen las transformaciones de los arrays de tipos enterossimilares. Los tipos similares son los siguientes: bool, char, uchar, puesto que estos tres tipos son enteros de un byte; los enteros de dos bytes short y ushort; los enteros de 4 bytes int, uint y color; long, ulong y datetime.

No hay duda que la correspondencia estricta es la mejor. Para conseguirla se puede usar las coversiones. El compilador no podrá con una situación de doble sentido, así que no hay que fiarse de las ligeras diferencias en los tipos y las transformaciones implícitas que hacen la función sobrecargada poco clara.

Si Usted tiene duda, recurra a las conversiones explícitas para conseguir la correspondencia estricta.

Como ejemplo de las funciones sobrecargadas en MQL5 puede servir el de la función ArrayInitialize().

Las reglas de sobrecarga de funciones también se aplican a la sobrecarga de métodos de clases.

La sobrecarga de las funciones del sistema está permitida pero en este caso hay que mirar que el compilador pueda elegir sin problema alguno la función necesaria. Como ejemplo, podemos sobrecargar la función del sistema fmax() de 3 formas distintas, pero sólo dos opciones van a ser correctas.

Ejemplo:

```
// sobrecarga está permitida - se diferencia por la cantidad de parámetros
double fmax(double a,double b,double c);

// sobrecarga con error
// la cantidad de parámetros es diferente, pero el último tiene el valor por defecto
// eso lleva a la ocultación de la función del sistema durante la llamada, pero esto
double fmax(double a,double b,double c=DBL_MIN);

// sobrecarga normal por el tipo de parámetro
int fmax(int a,int b);
```

Véase también

Sobrecarga, Funciones virtuales, Polimorfismo

Sobrecarga de operaciones

Para que la lectura y escritura del código sea más cómoda, se permite la sobrecarga de algunas operaciones. El operador de sobrecarga se escribe con la palabra clave operator. Está permitida la sobrecarga de las siguientes operaciones:

- binarias +,-,/,*,%,<<,>>,==,!=,<,>,<=,>=,=,+=,-=,/=,*=,%=,&=,|=,^=,<<=,>>=,&&,||,&,|,^;
- unarias +,-,++,--,!,~;
- operador de asignación =;
- operador de indexación [].

La sobrecarga de operaciones permite usar la notación operacional (anotación en forma de expresiones simples) con objetos complejos: estructuras y clases. La escritura de expresiones con el uso de las operaciones sobrecargadas facilita la percepción del código fuente, puesto que la implementación más compleja está oculta.

Como ejemplo vamos a considerar los números complejos, de amplio uso en las matemáticas, que se componen de la parte real e imaginaria. En el lenguaje MQL5 no hay un tipo de datos para representar los números complejos pero hay una posibilidad de crear un nuevo tipo de datos en forma de una estructura o clase. Vamos a declarar una estructura complex y definir dentro de ella cuatro métodos que realizan cuatro operaciones aritméticas:

```
//+------------------------------------------------------------------+
//| Estructura para operaciones con números complejos                |
//+------------------------------------------------------------------+
struct complex
  {
   double            re; // parte real
   double            im; // parte imaginaria
   //--- constructores
                     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) { }
   //--- operaciones aritméticas
   complex           Add(const complex &l,const complex &r) const;  // suma
   complex           Sub(const complex &l,const complex &r) const;  // resta
   complex           Mul(const complex &l,const complex &r) const;  // multiplicación
   complex           Div(const complex &l,const complex &r) const;  // división
  };
```

Ahora podemos declarar en nuestro código las variables que representan los números complejos, y trabajar con ellas.

Por ejemplo:

```
void OnStart()
  {
//--- declaramos e inicializamos las variables del tipo complejo
```

```
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
//--- sumamos dos números
   complex z;
   z=a.Add(a,b);
   PrintFormat("a+b=%.2f+i*%.2f",z.re,z.im);
//--- multiplicamos dos números
   z=a.Mul(a,b);
   PrintFormat("a*b=%.2f+i*%.2f",z.re,z.im);
//--- dividimos dos números
   z=a.Div(a,b);
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
  }
```

Pero para las operaciones aritméticas habituales con números complejos sería más cómodo utilizar los operadores habituales "+","-","*" y "/".

La palabra clave operator se utiliza para definir la función miembro que realiza la conversión del tipo. Las operaciones unarias y binarias para las variables-objetos de la clase pueden ser sobrecargadas como las funciones miembros no estáticas. Actúan de forma implícita sobre el objeto de la clase.

La mayoría de las operaciones binarias pueden ser sobrecargadas como funciones ordinarias que aceptan uno o ambos argumentos en forma de la variable de la clase o en forma del puntero al objeto de esta clase. Para nuestro tipo complex la sobrecarga en la declaración va a ser la siguiente:

```
   //--- operadores
   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)); }
```

Ejemplo completo del script:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declaramos e inicializamos las variables del tipo complejo
   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;
//--- sumamos dos números
   complex z=a+b;
   PrintFormat("a+b=%.2f+i*%.2f",z.re,z.im);
//--- multiplicamos dos números

   z=a*b;
   PrintFormat("a*b=%.2f+i*%.2f",z.re,z.im);
//--- dividimos dos números
   z=a/b;
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//---
  }
//+------------------------------------------------------------------+
//| Estructura para las operaciones con números complejos            |
//+------------------------------------------------------------------+
struct complex
  {
   double            re; // parte real
   double            im; // parte imaginaria
   //--- constructores
                     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) { }
   //--- operaciones aritméticas
   complex           Add(const complex &l,const complex &r) const;  // suma
   complex           Sub(const complex &l,const complex &r) const;  // resta
   complex           Mul(const complex &l,const complex &r) const;  // multiplicación
   complex           Div(const complex &l,const complex &r) const;  // división
   //--- operadores binarios
   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)); }
  };
//+------------------------------------------------------------------+
//|  Suma                                                            |
//+------------------------------------------------------------------+
complex complex::Add(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re+r.re;
   res.im=l.im+r.im;
//--- resultado
   return res;
  }
//+------------------------------------------------------------------+
//|   Resta                                                          |
```

```
//+------------------------------------------------------------------+
complex complex::Sub(const complex &l,const complex &r) const
  {
   complex res;
//---
   res.re=l.re-r.re;
   res.im=l.im-r.im;
//--- resultado
   return res;
  }
//+------------------------------------------------------------------+
//|   Multiplicación                                                 |
//+------------------------------------------------------------------+
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;
//--- resultado
   return res;
  }
//+------------------------------------------------------------------+
//|    División                                                      |
//+------------------------------------------------------------------+
complex complex::Div(const complex &l,const complex &r) const
  {
//--- número complejo vacío
   complex res(EMPTY_VALUE,EMPTY_VALUE);
//--- comprobación del cero
   if(r.re==0 && r.im==0)
     {
      Print(__FUNCTION__+": number is zero");
      return(res);
     }
//--- variables auxiliares
   double e;
   double f;
//--- selección del variante de cálculo
   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;
     }
//--- resultado
   return res;
  }
```

La mayoría de las operaciones unarias para las clases pueden ser sobrecargadas como funciones ordinarias que aceptan el único argumento objeto de la clase o puntero a él. Vamos a agregar la

sobrecarga de operaciones unarias "-" y "!".

```
//+------------------------------------------------------------------+
//| Estructura para las operaciones con números complejos            |
//+------------------------------------------------------------------+
struct complex
  {
   double            re;       // parte real
   double            im;       // parte imaginaria
...
   //--- operadores unarios
   complex operator-()  const; // menos unario
   bool    operator!()  const; // negación
  };
...
//+------------------------------------------------------------------+
//|   Sobrecarga del operador "menos unario"                         |
//+------------------------------------------------------------------+
complex complex::operator-() const
  {
   complex res;
//---
   res.re=-re;
   res.im=-im;
//--- resultado
   return res;
  }
//+------------------------------------------------------------------+
//|    Sobrecarga del operador "negación lógica"                     |
//+------------------------------------------------------------------+
bool complex::operator!() const
  {
//--- ¿es igual a cero la parte real e imaginaria del número complejo?
   return (re!=0 && im!=0);
  }
```

Ahora podemos comprobar el valor del número complejo respecto al cero y obtener valor negativo:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declaramos e inicializamos las variables del tipo complejo
   complex a(2,4),b(-4,-2);
   PrintFormat("a=%.2f+i*%.2f,   b=%.2f+i*%.2f",a.re,a.im,b.re,b.im);
//--- dividimos dos números
   z=a/b;
   PrintFormat("a/b=%.2f+i*%.2f",z.re,z.im);
//--- por defecto, el número complejo es igual a cero (en el constructor por defecto
   complex zero;
   Print("!zero=",!zero);
//--- asignamos valor negativo
   zero=-z;
   PrintFormat("z=%.2f+i*%.2f,  zero=%.2f+i*%.2f",z.re,z.im, zero.re,zero.im);
   PrintFormat("-zero=%.2f+i*%.2f",-zero.re,-zero.im);
//--- volvemos a comprobar la igualdad a cero
   Print("!zero=",!zero);
//---
  }
```

Fíjense que en este caso no hemos tenido la necesidad de sobrecargar la operación de asignación "=", porque las estructuras de tipos simples se puede copiar una a otra directamente. De esta manera, ahora podemos escribir el código para los cálculos que incluyen los números complejos en una manera a la que estamos acostumbrados.

La sobrecarga del operador de indexación permite obtener los valores de los arrays encerrados en un objeto de una manera más sencilla y habitual, y eso también contribuye a la mejor legibilidad y comprensión del código fuente de los programas. Por ejemplo, tenemos que asegurar el acceso a un símbolo en la cadena, según la posición especificada. Una cadena en el lenguaje MQL5 es un tipo separado string, que no es un array de símbolos. Pero mediante la operación de indexación sobrecargada podemos asegurar un trabajo sencillo y transparente en la clase creada CString:

```
//+------------------------------------------------------------------+
//| Clase para el acceso a los símbolos en la cadena como en el array de símbolos
//+------------------------------------------------------------------+
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()
  {
//--- array para recibir símbolos desde una cadena
   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;
//--- componemos una frase usando símbolos desde la variable str
   for(int i=0,n=ArraySize(x);i<n;i++)
     {
      res+=ShortToString(str[x[i]]);
     }
//--- mostramos resultado
   Print(res);
  }
```

Otro ejemplo de sobrecarga de la operación de indexación es el trabajo con matrices. Una matriz representa un array bidimensional dinámico, los tamaños de los arrays no están definidos de antemano. Por eso no se puede declarar un array de forma array[][] sin especificar el tamaño de la segunda dimensión y luego pasar este array como un parámetro. Como una posible solución puede ser una clase especial CMatrix que contiene un array de los objetos de la clase CRow.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- operaciones de adición y multiplicación de matrices
   CMatrix A(3),B(3),C();
//--- preparamos arrays para las cadenas
   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};
//--- llenamos matrices
   A[0]=a1; A[1]=a2; A[2]=a3;
   B[0]=b1; B[1]=b2; B[2]=b3;
//--- imprimimos las matrices en el diario "Asesores Expertos"
   Print("---- elementos de la matriz A");
   Print(A.String());
   Print("---- elementos de la matriz B");
   Print(B.String());

//--- suma de matrices
   Print("---- suma de matrices A y B");
   C=A+B;
//--- impresión de la representación string formateada
   Print(C.String());

//--- multiplicación de matrices
   Print("---- multiplicación de matrices A y B");
   C=A*B;
   Print(C.String());

//--- y ahora vamos a mostrar cómo obtener los valores en el estilo de los arrays din
   Print("Mostramos los valores de la matriz C elemento por elemento");
//--- repasamos en el ciclo las filas de la matriz - objetos CRow
   for(int i=0;i<3;i++)
     {
      string com="| ";
      //--- formamos filas desde la matriz para el valor
      for(int j=0;j<3;j++)
        {
         //--- obtenemos los elementos de la matriz por números de la fila y columna
         double element=C[i][j];// [i] - acceso CRow en el array m_rows[] ,
                                // [j] - operador de indexación sobrecargado en CRow
         com=com+StringFormat("a(%d,%d)=%G ; ",i,j,element);
        }
      com+="|";
      //--- imprimimos el valor de la fila
      Print(com);
     }
  }
//+------------------------------------------------------------------+
//|  Clase "Fila"                                                    |
//+------------------------------------------------------------------+
class CRow
  {
private:
   double            m_array[];
public:
   //--- constructores y destructor
                     CRow(void)          { ArrayResize(m_array,0);     }
                     CRow(const CRow &r) { this=r;                     }
                     CRow(const double &array[]);
```

```
                    ~CRow(void){};
   //--- número de elementos en la fila
   int                Size(void) const    { return(ArraySize(m_array));}
   //--- devuelve la cadena con valores
   string             String(void) const;
   //--- operador de indexación
   double             operator[](int i) const  { return(m_array[i]);   }
   //--- operadores de asignación
   void               operator=(const double  &array[]); // array
   void               operator=(const CRow & r);         // otro objeto CRow
   double             operator*(const CRow &o);          // objeto CRow para multiplic
  };
//+------------------------------------------------------------------+
//|  Constructor para inicializar una fila con un array              |
//+------------------------------------------------------------------+
void  CRow::CRow(const double &array[])
  {
   int size=ArraySize(array);
//--- si el array no está vacío
   if(size>0)
     {
      ArrayResize(m_array,size);
      //--- llenamos con valores
      for(int i=0;i<size;i++)
         m_array[i]=array[i];
     }
//---
  }
//+------------------------------------------------------------------+
//|  Operación de asignación para array                              |
//+------------------------------------------------------------------+
void CRow::operator=(const double &array[])
  {
   int size=ArraySize(array);
   if(size==0) return;
//--- llenamos array con valores
   ArrayResize(m_array,size);
   for(int i=0;i<size;i++) m_array[i]=array[i];
//---
  }
//+------------------------------------------------------------------+
//|  Operación de asignación para CRow                               |
//+------------------------------------------------------------------+
void CRow::operator=(const CRow  &r)
  {
   int size=r.Size();
   if(size==0) return;
//--- llenamos array con valores
   ArrayResize(m_array,size);
   for(int i=0;i<size;i++) m_array[i]=r[i];
//---
  }
//+------------------------------------------------------------------+
//|  Operador de multiplicación por otra fila                        |
//+------------------------------------------------------------------+
double CRow::operator*(const CRow &o)
  {
   double res=0;
//--- comprobaciones
   int size=Size();
   if(size!=o.Size() || size==0)
```

```
      {
       Print(__FUNCSIG__,": Error de multiplicación de dos matrices, sus tamaños no co
       return(res);
      }
//--- multiplicamos los arrays elemento por elemento y sumamos los productos
   for(int i=0;i<size;i++)
      res+=m_array[i]*o[i];
//--- resultado
   return(res);
  }
//+------------------------------------------------------------------+
//|  Devuelve la representaión string formateada                     |
//+------------------------------------------------------------------+
string CRow::String(void) const
  {
   string out="";
//--- si el tamaño del array es superior a cero
   int size=ArraySize(m_array);
//--- trabaja sólo con el número de elementos en el array superior a cero
   if(size>0)
     {
      out="{";
      for(int i=0;i<size;i++)
        {
         //--- reunimos los valores en la cadena
         out+=StringFormat(" %G;",m_array[i]);
        }
      out+=" }";
     }
//--- resultado
   return(out);
  }
//+------------------------------------------------------------------+
//|  Clase "Matriz"                                                  |
//+------------------------------------------------------------------+
class CMatrix
  {
private:
   CRow              m_rows[];

public:
   //--- constructores y destructor
                     CMatrix(void);
                     CMatrix(int rows)  { ArrayResize(m_rows,rows);             }
                    ~CMatrix(void){};
   //--- obtener tamaños de la matriz
   int               Rows()       const { return(ArraySize(m_rows));          }
   int               Cols()       const { return(Rows()>0? m_rows[0].Size():0); }
   //--- devuelve los valores de la columna en forma de una fila CRow
   CRow              GetColumnAsRow(const int col_index) const;
   //--- devuelve una cadena con los valores de la matriz
   string            String(void) const;
   //--- operador de indexación devuelve la cadena por su número
   CRow *operator[](int i) const        { return(GetPointer(m_rows[i]));      }
   //--- operador de adición
   CMatrix           operator+(const CMatrix &m);
   //--- operador de multiplicación
   CMatrix           operator*(const CMatrix &m);
   //--- operador de asignación
   CMatrix          *operator=(const CMatrix &m);
  };
```

```
//+------------------------------------------------------------------+
//|  Un constructor predefinido, crea un array de filas con el tamaño cero |
//+------------------------------------------------------------------+
CMatrix::CMatrix(void)
  {
//--- número de filas cero en la matriz
   ArrayResize(m_rows,0);
//---
  }
//+------------------------------------------------------------------+
//|  Devuelve los valores de la columna en forma de la fila CRow                  |
//+------------------------------------------------------------------+
CRow  CMatrix::GetColumnAsRow(const int col_index) const
  {
//--- una variable para recibir valores desde la columna
   CRow row();
//--- número de filas en la matriz
   int rows=Rows();
//--- si el número de filas es mayor a cero, ejecutamos la operación
   if(rows>0)
     {
      //--- array para recibir valores de la columna con el índice col_index
      double array[];
      ArrayResize(array,rows);
      //--- llenando array
      for(int i=0;i<rows;i++)
        {
         //--- comprobación del número de la columna para la fila i para ver si sale
         if(col_index>=this[i].Size())
           {
            Print(__FUNCSIG__,": ¡Error! El número de la columna es ",col_index,"> de
            break; // row se queda como un objeto no inicializado
           }
         array[i]=this[i][col_index];
        }
      //--- creamos la fila CRow a base de los valores del array
      row=array;
     }
//--- resultado
   return(row);
  }
//+------------------------------------------------------------------+
//|  Suma de dos matrices                                            |
//+------------------------------------------------------------------+
CMatrix CMatrix::operator+(const CMatrix &m)
  {
//--- número de filas y columnas en la matriz pasada
   int cols=m.Cols();
   int rows=m.Rows();
//--- matriz para recibir el resultado de adición
   CMatrix res(rows);
//--- los tamaños de la matriz deben coincidir
   if(cols!=Cols() || rows!=Rows())
     {
      //--- no se puede sumar
      Print(__FUNCSIG__,": Error de adición de dos matrices, los tamaños no coinciden
      return(res);
     }
//--- array auxiliar
   double arr[];
   ArrayResize(arr,cols);
```

```
//--- repasamos las filas para sumar
   for(int i=0;i<rows;i++)
     {
      //--- escribimos los resultados de adición de las cadenas de las matrices en el
      for(int k=0;k<cols;k++)
        {
         arr[k]=this[i][k]+m[i][k];
        }
      //--- colocamos el array en la fila de la matriz
      res[i]=arr;
     }
//--- devolvemos el resultado de adición de matrices
   return(res);
  }
//+------------------------------------------------------------------+
//|  Multiplicación de dos matrices                                  |
//+------------------------------------------------------------------+
CMatrix CMatrix::operator*(const CMatrix &m)
  {
//--- número de columnas de la primera matriz, número de filas en la matriz pasada
   int cols1=Cols();
   int rows2=m.Rows();
   int rows1=Rows();
   int cols2=m.Cols();
//--- matriz para recibir el resultado de multiplicación
   CMatrix res(rows1);
//--- las matrices tiene que ser compatibles
   if(cols1!=rows2)
     {
      //--- no se puede multiplicar
      Print(__FUNCSIG__,": Error de multiplicación de dos matrices, el formato es inc
            "- el número de columnas en el primer factor debe ser igual al número de
      return(res);
     }
//--- array auxiliar
   double arr[];
   ArrayResize(arr,cols1);
//--- llenamos filas en la matriz de multiplicación
   for(int i=0;i<rows1;i++)// repasamos filas
     {
      //--- ponemos a cero el array receptor
      ArrayInitialize(arr,0);
      //--- repasamos elementos en la fila
      for(int k=0;k<cols1;k++)
        {
         //--- cogeremos desde la matriz m los valores de la columna k en forma de la
         CRow column=m.GetColumnAsRow(k);
         //--- multiplicamos dos filas y escribimos el resultado de la multiplicación
         arr[k]=this[i]*column;
        }
      //--- colocamos el array arr[] en la fila i de la matriz
      res[i]=arr;
     }
//--- devolvemos el producto de dos matrices
   return(res);
  }
//+------------------------------------------------------------------+
//|  Operación de asignación                                         |
//+------------------------------------------------------------------+
CMatrix *CMatrix::operator=(const CMatrix &m)
  {
```

```
//--- encontramos y fijamos el número de filas
   int rows=m.Rows();
   ArrayResize(m_rows,rows);
//--- llenamos nuestras filas con los valores de las filas de matriz pasada
   for(int i=0;i<rows;i++) this[i]=m[i];
//---
   return(GetPointer(this));
  }
//+------------------------------------------------------------------+
//|  Representación string de la matriz                              |
//+------------------------------------------------------------------+
string CMatrix::String(void) const
  {
   string out="";
   int rows=Rows();
//--- formamos cadena por cadena
   for(int i=0;i<rows;i++)
     {
      out=out+this[i].String()+"\r\n";
     }
//--- resultado
   return(out);
  }
```

Véase también

Sobrecarga, Operaciones aritméticas, Sobrecarga de funciones, Prioridades y orden de las operaciones

Descripción de funciones externas

El tipo de funciones externas definidas en otro módulo, tiene que ser explícitamente descrito. La falta de esta descripción puede llevar a los errores en la fase de compilación, composición o ejecución del programa. Durante la descripción de un objeto externo, use la palabra clave *#import* indicando el módulo.

Ejemplos:

```
#import "user32.dll"
  int     MessageBoxW(int hWnd ,string szText,string szCaption,int nType);
  int     SendMessageW(int hWnd,int Msg,int wParam,int lParam);
#import "lib.ex5"
  double  round(double value);
#import
```

Mediante la importación es muy fácil describir las funciones invocadas de las DLL externas o de las bibliotecas compiladas EX5. Las bibliotecas EX5 son unos archivos ex-5 que tienen la propiedad de library. Sólo las funciones descritas con el modificador export pueden ser importadas de las bibliotecas EX5.

Véase también

Sobrecarga, Funciones virtuales, Polimorfismo

Exportación de funciones

Hay posibilidad de usar en el programa MQL5 la función que ha sido declarada en otro programa MQL5 con el postmodificador *export*. Esta función se llama exportable, y puede invocarse de otros programas después de la compilación.

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

Este modificador indica al compilador que inserte la función en la tabla de funciones EX5 que serán exportadas por el archivo ejecutor ex5. Sólo las funciones con este modificador serán accesibles ("visibles") desde otros programas mql5.

La propiedad library indica al compilador que este archivo EX5 va a ser una biblioteca, y el compilador lo pondrá en la cabecera de EX5.

Todas las funciones planificadas como exportables deben ser marcadas con el modificador *export*.

Véase también

Sobrecarga, Funciones virtuales, Polimorfismo

Funciones de procesamiento de eventos

En el lenguaje MQL5 está previsto el procesamiento de algunos eventos predefinidos. Las funciones que manejan estos eventos tienen que ser definidos en el programa MQL5; el nombre de la función, el tipo de valor devuelto, composición de parámetros (si éstos existen) y sus tipos tienen que corresponder estrictamente a la descripción de la función de procesamiento de eventos.

El manejador de eventos del terminal de cliente identifica las funciones, que procesan uno u otro evento, precisamente por el tipo de valor devuelto y por los tipos de parámetros. La función no se usa para manejar los eventos, si en ésta están indicados otros parámetros que no corresponden a las descripciones siguientes, o está indicado otro tipo de valor devuelto.

OnStart

La función OnStart() es manejador de evento Start, que se genera automáticamente **sólo** para los **scripts** a ser ejecutados. Esta función debe ser del tipo void, sin tener parámetros:

```
void OnStart();
```

Para la función OnStart() se admite especificar el tipo de valor devuelto int.

OnInit

La función OnInit() es el manejador de evento Init. Puede tener el tipo void o int, no tiene parámetros:

```
void OnInit();
```

El evento Init se genera justo después de haberse cargado un Asesor Experto o un indicador; este evento no se genera para los scripts. La función OnInit() se usa para la inicialización. Si OnInit() tiene el valor devuelto del tipo int, entonces el código no nulo de la devolución significa una inicialización fallida y genera el evento Deinit con el código de la causa de deinicialización REASON_INITFAILED.

Para la optimización de los parámetros de entrada del EA se recomienda utilizar los valores de la enumeración ENUM_INIT_RETCODE como el código de devolución. Estos valores sirven para organizar el control del proceso de optimización, incluyendo la selección de los agentes de pruebas más apropiados. Durante la inicialización del EA, antes de iniciar el mismo proceso de simulación, se puede solicitar la información sobre la configuración y los recursos del agente (número de núcleos, volumen de la memoria libre, etc.) utilizando la función TerminalInfoInteger(). Y basándose en la información recibida permitir el uso de este agente, o bien rechazarlo para la optimización de este EA.

ENUM_INIT_RETCODE

Identificador	Descripción
INIT_SUCCEEDED	La inicialización se ha finalizado con éxito. Se puede seguir con la simulación del EA. Este código significa lo mismo que el valor nulo - la inicialización del EA en el Probador ha pasado con éxito.
INIT_FAILED	Inicialización fallida. No merece la pena continuar la simulación debido a los errores

	insuperables. Por ejemplo, no se ha podido crear el indicador necesario para el funcionamiento del EA. La devolución de este valor significa lo mismo que la devolución de un valor distinto del cero. Significa que la inicialización del EA en el Probador no ha tenido éxito. La simulación para este conjunto de parámetros del EA no va a llevarse a cabo, el agente está disponible para recibir una nueva tarea.
INIT_PARAMETERS_INCORRECT	Se utiliza por el programador para marcar un conjunto incorrecto de parámetros de entrada. La cadena del resultado que contiene este código de retorno se colorea con el rojo en la tabla general de optimización. Cuando el Probador de Estrategias recibe este valor, nunca va a pasar esta tarea a otros agentes para que vuelvan a ejecutarlo.
INIT_AGENT_NOT_SUITABLE	Durante la inicialización no ha surgido ningún error en el funcionamiento del programa, pero por alguna razón este agente no vale para hacer la prueba. Por ejemplo, no hay suficiente memoria operativa, no hay soporte OpenCL, etc. Después de la devolución de este código, el agente ya no va a recibir tareas hasta que se finalice esta optimización.

La función OnInit() del tipo void siempre indica a una inicialización exitosa.

OnDeinit

La función OnDeinit() se llama durante la deinicialización y juega papel de manejador de evento Deinit. Tiene que ser declarada con el tipo void y tener un parámetro del tipo const int que contiene el código de la causa de deinicialización. Si está declarado otro tipo, el compilador lanzará un aviso pero la función no será invocada. Para los scripts el evento Deinit no se genera, y por tanto, en éstos no se puede usar la función OnDeinit().

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

El evento Deinit se genera para los Asesores Expertos e indicadores en las siguientes ocasiones:

- antes de la reinicialización debido al cambio de símbolo o período del gráfico al cual el programa mql5 es atado;
- antes de la reinicialización debido al cambio de los parámetros de entrada;
- antes de descargar un programa mql5.

OnTick

El evento NewTick se genera **únicamente para los Asesores Expertos** cuando se recibe un nuevo tick para el símbolo, al diagrama del cual está atado el Asesor. Es inútil determinar la función OnTick() en un indicador personalizado o en un script, porque para ellos el evento Tick no se genera.

El evento NewTick se genera sólo para los Asesores Expertos pero esto no significa que ellos tienen que tener la función OnTick() de una forma obligatoria, porque para los Asesores se generan no sólo los eventos NewTick, sino también los eventos Timer, BookEvent y ChartEvent. Tiene que ser declarada con el tipo void, no tiene parámetros:

```
void OnTick();
```

OnTimer

La función OnTimer() se llama cuando se inicia el evento Timer que se genera por el temporizador del sistema sólo para los asesores e indicadores; no se puede usarla en los scripts. La frecuencia del inicio de este evento se establece cuando la función EventSetTimer() efectua la suscripción para obtener avisos sobre el evento Timer.

Para dar de baja dicha suscripción a recibir los eventos del temporizador para un Asesor en concreto se utiliza la función EventKillTimer(). La función tiene que ser declarada con el tipo void, no tiene parámetros:

```
void OnTimer();
```

Se recomienda llamar a la función EventSetTimer() una sola vez en la función OnInit(), y también una sola vez EventKillTimer() en OnDeinit().

Cada Asesor Experto y cada indicador trabaja con su propio temporizador y recibe eventos sólo de él. Una vez terminada la sesión del programa mql5, el temporizador se elimina de una manera forzosa, si ha sido creado pero no ha sido desactivado por la función EventKillTimer().

OnTrade

La función se llama cuando se inicia el evento Trade, que aparece si se cambia la lista de las órdenes presentadas y posiciones abiertas, historial de órdenes y historial de transacciones. Cuando cualquier operación comercial (presentación de orden pendiente, apertura/cierre de posición, establecimiento de stops, accionamiento de órdenes pendientes, etc.) se efectua de una manera correspondiente, el historial de órdenes y transacciones y/o la lista de posiciones y órdenes corrientes se cambian.

```
void OnTrade();
```

Al recibir este evento (si esto requieren las condiciones de la estrategia comercial), el mismo usuario debe comprobar en el código el estado de la cuenta. Si la llamada a la función OrderSend() se ha realizado con éxito y ha devuelto el valor true, esto significa que el servidor comercial ha colocado la orden en la cola para ser ejecutado y le ha asignado un número de ticket. En cuanto el servidor procese esta orden, el evento Trade será generado. Y si el usuario ha memorizado el valor del ticket, durante el procesamiento de evento OnTrade() podrá averiguar con su ayuda qué es lo que haya pasado exactamente con la orden.

OnTradeTransaction

Como resultado de ejecución de ciertas acciones con la cuenta de trading su estado se cambia. A estas acciones les pertenecen:

- El envío de una solicitud comercial por parte de cualquier aplicación MQL5 en el terminal de cliente utilizando la función OrderSend y OrderSendAsync, con su posterior ejecución;
- El envío de una solicitud comercial a través de la interfaz gráfica del terminal y su posterior ejecución;
- El accionamiento de órdenes pendientes y órdenes Stop en el servidor;
- La ejecución de operaciones en el servidor de trading.

Como resultado de estas acciones, para la cuenta se ejecutan las transacciones comerciales:

- tramitación de la solicitud comercial;
- cambio de órdenes abiertas;
- cambio del historial de órdenes;
- cambio del historial de operaciones;
- cambio de posiciones.

Por ejemplo, al enviar una orden de compra, ésta se tramita, para la cuenta se crea una orden de compra correspondiente, se realiza la ejecución de la orden, su eliminación de la lista de las abiertas, se agrega al historial de órdenes, luego la operación correspondiente se agrega al historial, y se crea una posición nueva. Pues todas estas acciones son transacciones comerciales. La llegada de cada una de estas transacciones al terminal es el evento TradeTransaction. Este evento llama al manejador OnTradeTransaction

```
void  OnTradeTransaction(
   MqlTradeTransaction&    trans,        // estructura de transacción comercial
   MqlTradeRequest&        request,      // estructura de solicitud
   MqlTradeResult&         result        // estructura de respuesta
   );
```

El manejador contiene tres parámetros:

- **trans** - este parámetro obtiene la estructura MqlTradeTransaction que describe la transacción comercial aplicada a la cuenta de trading;
- **request** - este parámetro obtiene la estructura MqlTradeRequest que describe la solicitud comercial;
- **result** - este parámetro obtiene la estructura MqlTradeResult que describe el resultado de ejecución de la solicitud comercial.

Los dos últimos parámetros **request** y **result** se llenan con los valores sólo para la transacción del tipo TRADE_TRANSACTION_REQUEST, la información sobre el parámetro se puede obtener del parámetro *type* de la variable **trans**. Fíjense que en este caso el campo *request_id* en la variable **result** contiene el identificador de la solicitud comercial **request** cuya ejecución ha provocado la aparición de la transacción comercial descrita en la variable **trans**. La presencia del identificador de la solicitud permite vincular la acción ejecutada (llamada a la función OrderSend o OrderSendAsync) con el resultado de esta acción que se traspasa en OnTradeTransaction().

Una solicitud comercial enviada desde el terminal manualmente o a través de las funciones de trading OrderSend()/OrderSendAsync() puede ocasionar en el servidor de trading varias transacciones consecutivas. Entendiéndose que el orden de llegada de estas transacciones al terminal no se garantiza, por eso no se puede construir su algoritmo de trading esperando la llegada de unas transacciones comerciales tras la llegada de otras. Además, las transacciones pueden perderse por el

camino del servidor al terminal.

- Todos los tipos de transacciones comerciales se describen en la enumeración ENUM_TRADE_TRANSACTION_TYPE.
- La estructura MqlTradeTransaction que describe la transacción comercial se llena de una manera diferente en función del tipo de transacción. Por ejemplo, para las transacciones del tipo TRADE_TRANSACTION_REQUEST hay que analizar sólo un campo - type (tipo de la transacción comercial). Para obtener la información adicional hay que analizar el segundo y el tercer parámetro de la función OnTradeTransaction (request y result). Más detalles se puede encontrar en el apartado "Estructura de transacción comercial".
- No toda la información disponible sobre las órdenes, operaciones y posiciones (por ejemplo, un comentario) se pasa en la descripción de una transacción comercial. Para obtener la información más detallada, hay que usar las funciones OrderGet*, HistoryOrderGet*, HistoryDealGet* y PositionGet*.

Una vez aplicadas las transacciones comerciales a la cuenta de cliente, ellas se colocan sucesivamente a la cola de transacciones comerciales del terminal, de donde se pasan sucesivamente al punto de entrada OnTradeTransaction en orden de su llegada al terminal.

Mientras que el EA procese las transacciones comerciales con el manejador OnTradeTransaction, el terminal sigue procesando las transacciones que vayan llegando. De esta manera, el estado de la cuenta de trading ya puede cambiarse durante el proceso de trabajo del OnTradeTransaction. Por ejemplo, mientras que el programa MQL5 esté procesando el evento de agregación de una orden nueva, ésta puede ser ejecutada, eliminada de la lista de las abiertas y pasada al historial. A continuación, el programa será avisado sobre todos estos eventos.

La longitud de la cola de transacciones es de 1024 elementos. Si OnTradeTransaction va a tardar en procesar la transacción de turno, las transacciones antiguas pueden ser expulsadas por las nuevas.

- Por lo general, no existe la proporción exacta respecto al número de llamadas a OnTrade y a OnTradeTransaction. Una llamada a OnTrade corresponde a una o varias llamadas a OnTradeTransaction.
- OnTrade se invoca tras las llamadas correspondientes a OnTradeTransaction.

OnTester

La función OnTester() es el manejador del evento Tester que se genera automáticamente una vez terminado el chequeo histórico del Asesor Experto en el intervalo de datos especificado. La función tiene que ser declarada con el tipo double, no tiene parámetros:

```
double OnTester();
```

La función se invoca justamente antes de la llamada a la función OnDeinit() y tiene el tipo del valor devuelto double. La función OnTester() puede ser usada solamente en los Asesores Expertos durante el chequeo. En primer lugar está destinada para el cálculo de un valor que se usa como criterio Custom max durante la optimización genética de los parámetros de entrada.

Durante la optimización genética la selección de resultados dentro de una generación se realiza en orden descendiente. Es decir, desde el punto de vista del criterio de optimización, los mejores resultados son los que tienen el mayor valor (para el criterio de optimización Custom max se toman en

cuenta los valores devueltos por la función OnTester). Con este tipo de selección los peores valores se colocan al final, y luego no toman parte en la formación de la siguiente generación.

OnTesterInit

La función OnTesterInit() es el manejador del evento TesterInit que se genera automáticamente antes de iniciar la optimización del EA en el Probador de Estrategias. La función tiene que ser definida con el tipo void. No tiene parámetros:

```
void OnTesterInit();
```

El EA que cuenta con el manejador OnTesterDeinit() o OnTesterPass(), al iniciarse la optimización, se carga automáticamente en un gráfico separado con el símbolo y período especificados en el Probador y recibe el evento TesterInit. Esta función se utiliza para inicializar el EA antes del inicio de la optimización para el posterior procesamiento de los resultados de la optimización.

OnTesterPass

La función OnTesterPass() es el manejador del evento TesterPass que se genera automáticamente cuando llega un frame durante la optimización del EA en el Probador de Estrategias. La función tiene que ser definida con el tipo void. No tiene parámetros:

```
void OnTesterPass();
```

El EA con el manejador OnTesterPass() se carga automáticamente en un gráfico nuevo del terminal con el símbolo/período especificados para la simulación, y recibe durante la optimización los eventos TesterPass cuando llegue un frame. La función está destinada para el procesamiento dinámico de los resultados de la optimización directamente "al vuelo", sin esperar su finalización. La agregación de los frames se realiza por la función FrameAdd(), que puede ser invocada cuando se finaliza el repaso único en el manejador OnTester().

OnTesterDeinit

La función OnTesterDeinit() es el manejador del evento TesterDeinit que se genera automáticamente tras finalizarse la optimización del EA en el Probador de Estrategias. La función tiene que ser definida con el tipo void. No tiene parámetros:

```
void OnTesterDeinit();
```

El EA con el manejador TesterDeinit() se carga automáticamente en el gráfico al iniciarse la optimización, y recibe el evento TesterDeinit tras su finalización. Esta función está destinada para el procesamiento final de todos los resultados de la optimización.

OnBookEvent

La función OnBookEvent() es manejador de evento BookEvent. El evento BookEvent se genera sólo para los Asesores si se cambia el estado de la profundidad de mercado (Depth of Market). Debe tener el tipo void y un parámetro del tipo string:

```
void OnBookEvent (const string& symbol);
```

Para recibir los eventos BookEvent por cualquier símbolo, es suficiente suscribirse previamente a la recepción de estos eventos para este símbolo mediante la función MarketBookAdd(). Para dar de baja la recepción del evento BookEvent por un símbolo concreto, es necesario llamar a la función

MarketBookRelease().

A diferencia de otros eventos, el evento BookEvent es de difusión. Eso significa que si un Asesor Experto se suscribe a la recepción del evento BookEvent a través de la función MarketBookAdd, todos los demás Asesores que tienen el manejador OnBookEvent() van a recibir este evento. Por eso es necesario analizar el nombre del símbolo que se traspasa al manejador en calidad del parámetro *const string& symbol*.

OnChartEvent

OnChartEvent() es manejador del grupo de eventos ChartEvent:

- CHARTEVENT_KEYDOWN — evento de pulsación de teclas del teclado cuando la ventana del gráfico está en el foco;
- CHARTEVENT_MOUSE_MOVE — eventos de mover el ratón y pulsar botones del ratón (si para el gráfico está establecida la propiedad CHART_EVENT_MOUSE_MOVE=true);
- CHARTEVENT_OBJECT_CREATE — evento de creación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_CREATE=true);
- CHARTEVENT_OBJECT_CHANGE — evento de cambio de propiedades de un objeto a través del diálogo de propiedades;
- CHARTEVENT_OBJECT_DELETE — evento de eliminación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_DELETE=true);
- CHARTEVENT_OBJECT_CLICK — evento de cliqueo con el ratón sobre el gráfico;
- CHARTEVENT_OBJECT_DRAG — evento de movimiento de un objeto gráfico mediante el ratón;
- CHARTEVENT_OBJECT_ENDEDIT — evento del fin de edición de texto en el campo de introducción de un objeto gráfico LabelEdit;
- CHARTEVENT_CHART_CHANGE — evento de modificación del gráfico;
- CHARTEVENT_CUSTOM+n — identificador del evento de usuario donde la n se encuentra en el rango de 0 a 65535.
- CHARTEVENT_CUSTOM_LAST — el último identificador aceptable del evento de usuario (CHARTEVENT_CUSTOM+65535).

La función puede invocarse sólo en los Asesores e indicadores. Tiene que poseer el tipo void y 4 parámetros:

```
void OnChartEvent(const int id,         // identificador de evento
                  const long& lparam,   // parámetro de evento del tipo long
                  const double& dparam, // parámetro de evento del tipo double
                  const string& sparam  // parámetro de evento del tipo string
  );
```

Para cada tipo de evento, los parámetros de entrada de la función OnChartEvent() tienen los determinados valores que son necesarios para procesar este evento. En la tabla de abajo se enumeran los eventos y valores pasados como parámetros.

Evento	Valor del parámetro id	Valor del parámetro lparam	Valor del parámetro dparam	Valor del parámetro sparam

Evento del teclado	CHARTEVENT_KEYDOWN	código de la tecla pulsada	Número de pulsaciones de la tecla generadas mientras ésta se mantenía en estado pulsado	Valor literal de la máscara de bits que describe el estatus de las teclas del teclado
Eventos del ratón (si para el gráfico está establecida la propiedad CHART_EVENT_MOUSE_MOVE=true)	CHARTEVENT_MOUSE_MOVE	coordinada X	coordinada Y	Valor literal de la máscara de bits que describe el estatus de los botones del ratón
Evento de creación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_CREATE=true)	CHARTEVENT_OBJECT_CREATE	—	—	Nombre del objeto gráfico creado
Evento de cambio de propiedades de un objeto a través del diálogo de propiedades	CHARTEVENT_OBJECT_CHANGE	—	—	Nombre del objeto gráfico modificado
Evento de eliminación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_DELETE=true)	CHARTEVENT_OBJECT_DELETE	—	—	Nombre del objeto gráfico eliminado
Evento de cliquear sobre un gráfico	CHARTEVENT_CLICK	coordenada X	coordenada Y	—
Evento de cliquear sobre un objeto gráfico	CHARTEVENT_OBJECT_CLICK	coordenada X	coordenada Y	Nombre del objeto gráfico en el que ha

				ocurrido un evento
Evento de mover un objeto gráfico con el ratón	CHARTEVENT_OBJECT_DRAG	—	—	Nombre del objeto gráfico movido
Evento del fin de edición del texto en el campo de introducción del objeto gráfico	CHARTEVENT_OBJECT_ENDEDIT	—	—	Nombre del objeto gráfico "Campo de texto" donde se ha finalizado la introducción del texto
Evento de modificación del gráfico	CHARTEVENT_CHART_CHANGE	—	—	—
Identificador del evento de usuario	CHARTEVENT_CUSTOM+N	Valor determinado por la función EventChartCustom()	Valor determinado por la función EventChartCustom()	Valor determinado por la función EventChartCustom()

OnCalculate

La función OnCalculate() se invoca sólo en los indicadores si surge la necesidad de calcular los valores de indicador por evento Calculate. Habitualmente eso ocurre cuando se recibe un nuevo tick por símbolo para el que se calcula el indicador. Además, no es obligatorio que el indicador esté anclado a un gráfico de precio del dicho símbolo.

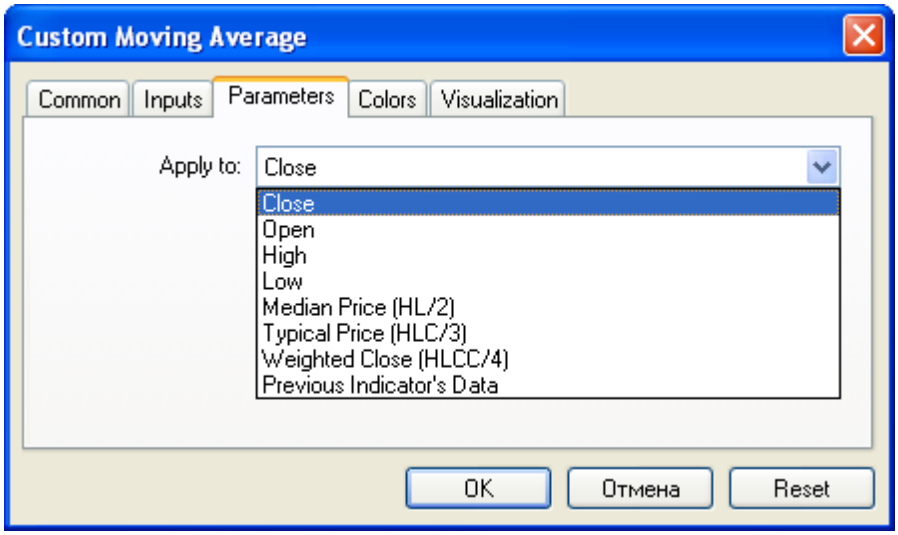
La función OnCalculate() debe tener el tipo de valor devuelto int. Existen dos posibilidades de definición. No se puede usar las dos opciones de la función dentro un indicador.

La primera forma de la llamada está destinada para los indicadores que pueden ser calculados sobre un buffer de datos. El ejemplo de este indicador es Custom Moving Average.

```
int OnCalculate (const int rates_total,      // tamaño del array price[]
                 const int prev_calculated,  // cantidad procesada de barras en la an
                 const int begin,            // de donde se empiezan los datos signif
                 const double& price[]       // array a calcular
   );
```

Como array price[] puede ser pasada una de las series temporales de precios o calculado el búfer de algún indicador. Para determinar la dirección de indexación en el array price[] es necesario invocar la función ArrayGetAsSeries(). Para no depender de los valores predefinidos es necesario invocar incondicionalmente la función ArraySetAsSeries() para los arrays con los que se prevé trabajar.

La elección del indicador o serie temporal apropiados en calidad del array price[] se realiza por el usuario a la hora de iniciar el indicador en la pestaña "Parameters". Para eso se necesita indicar el elemento necesario en la lista desplegable del campo "Apply to".



Par obtener valores del indicador personalizado desde otros programas mql5, se usa la función iCustom() que devuelve el manejador (handle) del indicador para la siguiente operación. Con eso también se puede indicar el array necesario price[] o manejador (handle) de otro indicador. Este parámetro tiene que ser pasado el último en la lista de las variables de entrada del indicador personalizado.

Ejemplo:

```
void OnStart()
  {
//---
   string terminal_path=TerminalInfoString(STATUS_TERMINAL_PATH);
   int handle_customMA=iCustom(Symbol(),PERIOD_CURRENT, "Custom Moving Average",13,0,
   if(handle_customMA>0)
      Print("handle_customMA = ",handle_customMA);
   else
      Print("Cannot open or not EX5 file '"+terminal_path+"\\MQL5\\Indicators\\"+"Cus
  }
```

En este ejemplo el último parámetro pasa el valor PRICE_TYPICAL (desde la enumeración ENUM_APPLIED_PRICE) que indica que el indicador personalizado va a ser construido por los precios típicos recibidos como (High+Low+Close)/3. Si el parámetro no se indica, el indicador se construye por los valores PRICE_CLOSE, es decir, por los precios de cierre de cada barra.

Otro ejemplo que demuestra el traspaso del manejador (handle) de indicador por el último parámetro para especificar el array price[], se muestra en la descripción de la función iCustom().

La segunda forma de la llamada sirve para todos los demás indicadores en los cuales para el cálculo se utiliza más de una serie temporal.

```
int OnCalculate (const int rates_total,      // tamaño de series temporales de entrad
                 const int prev_calculated,  // procesado de barras en la anterior ll
                 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
   );
```

Los parámetros open[], high[], low[] y close[] contienen los arrays con los precios de apertura, precio máximo, mínimo y precios de cierre del período en curso. El parámetro time[] contiene el array con valores de apertura, el parámetro spread[] es el array que contiene el historial de spreads (si spread está previsto para este instrumento comercial). Los parámetros volume[] y tick_volume[] contienen respectivamente el historial del volumen comercial y del volumen de tick.

Para determinar la dirección de indexación dentro de los arrays time[], open[], high[], low[], close[], tick_volume[], volume[] y spread[] es necesario llamar a la función ArrayGetAsSeries(). Para no depender de los valores predefinidos es necesario invocar incondicionalmente la función ArraySetAsSeries() para los arrays con los que se prevé trabajar.

El primer parámetro rates_total contiene la cantidad de barras disponibles para el indicador para ser calculados, y corresponde a la cantidad de barras disponibles en el gráfico.

Cabe destacar el vínculo entre el valor de la función devuelta OnCalculate() y el segundo parámetro de entrada prev_calculated. Cuando se invoca la función el parámetro prev_calculated contiene el valor que la función OnCalculate() **ha devuelto** durante la llamada **anterior**. Esto permite realizar los algoritmos económicos de cálculo del indicador personalizado con el fin de evitar los cálculos repetidos para las barras que no se hayan cambiado desde el arranque anterior de esta función.

Para eso es suficiente devolver el valor del parámetro rates_total que contiene la cantidad de barras durante la corriente llamada a la función. Si desde la última llamada a la función OnCalculate() los datos de precios han sido cambiados (se ha cargado un historial más profunda o los blancos en el historial han sido llenados), el mismo terminal pondrá el valor cero al parámetro entrante prev_calculated.

Nota: si la función OnCalculate devuelve el valor cero, los valores del indicador no se mostrarán en la ventana DataWindow del terminal de cliente.

Para el mejor entendimiento, será útil iniciar el indicador cuyo código se encuentra más abajo.

Ejemplo de indicador:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   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[])
  {
//--- obtendremos una cantidad de barras disponibles para el símbolo y período actual
   int bars=Bars(Symbol(),0);
   Print("Bars = ",bars,", rates_total = ",rates_total,", prev_calculated = ",prev_ca
   Print("time[0] = ",time[0]," time[rates_total-1] = ",time[rates_total-1]);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

Véase también

Ejecución del programa, Eventos del terminal de usuario, Trabajo con eventos

Variables

Declaración de variables

Las variables tienen que estar declaradas antes de empezar a usarlas. Para identificar las variables se utilizan los nombres únicos. Las descripciones de variables se utilizan para su determinación y declaración de tipos. Las descripciones no son unos operadores.

Los tipos simples son:

- char, short, int, long, uchar, ushort, uint, ulong – números enteros;
- color – número entero que representa el color RGB;
- datetime – fecha y hora es un número sin signo que contiene la cantidad de segundos pasados desde 0 horas del 1 de enero de 1970;
- bool – valores lógicos true y false;
- double – números de doble precisión en punto flotante;
- float – números de precisión sencilla en punto flotante;
- string – cadenas de caracteres.

Ejemplos:

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

Tipos complejos o compuestos:

Las estructuras son unos tipos de datos compuestos que se construyen con la ayuda de otros tipos.

```
struct MyTime
  {
   int hour;    // 0-23
   int minute;  // 0-59
   int second;  // 0-59
  };
...
MyTime strTime; // variable del tipo de la estructura MyTime declarada anteriormente
```

No se puede declarar las variables del tipo de estructuras hasta que la estructura esté declarada.

Arrays

El array es un conjunto indexado de datos del mismo tipo:

```
int    a[50];       // Array unidimensional de 50 números enteros.
double m[7][50];    // Array bidimensional compuesto por siete arrays,
                    // cada uno de los cuales contiene 50 números.
```

```
MyTime t[100];      // array que contiene elementos del tipo MyTime
```

Sólo un número entero puede ser el índice del array. Se admiten los arrays que no tienen más de cuatro dimensiones. La numeración de los elementos de un array se empieza con 0. El último elemento del array unidimensional tiene el número a 1 menos que el tamaño del array, es decir, la llamada al último elemento del array de 50 números enteros se verá de la siguiente manera a[49]. Lo mismo se puede decir de los arrays multidimensionales, la indexación de una dimensión se realiza de 0 al tamaño de dimensión -1. El último elemento del array bidimensional del ejemplo aparece como m[6][49].

Los arrays estáticos no pueden ser representados como series temporales, es decir, no se les puede aplicar la función ArraySetAsSeries(), que establece el acceso a los elementos del array desde el fin del array a su principio. Si es necesario establecer el acceso al array como en series temporales, hay que usar el objeto del array dinámico.

Al acceder fuera de los límites del array, el subsistema ejecutivo generará un error crítico y la ejecución del programa se detendrá.

Especificadores de acceso

Los especificadores de acceso indican al compilador cómo se puede realizar el acceso a las variables, elementos de las estructuras o de las clases.

El especificador const declara una variable como una constante y no permite cambiar el valor de esta variable durante el proceso de ejecución del programa. La inicialización de la variable durante su declaración se permite sólo una vez. El especificador const no puede ser aplicado a los elementos de estructuras y clases.

Ejemplo

```
int OnCalculate (const int rates_total,      // tamaño del array price[]
                 const int prev_calculated,  // barras procesadas durante la llamada
                 const int begin,            // de donde se empiezan los datos signif
                 const double& price[]       // array para el cálculo
   );
```

Para acceder a los elementos de las estructuras y clases se utilizan los siguientes especificadores:

- public - no limita el acceso con nada a la variable o al método de clase;
- protected - permite el acceso desde los métodos de la misma clase, y también desde los métodos de las clases que se heredan abiertamente. Otro acceso es imposible;
- private - permite el acceso a las variables y métodos de clase únicamente desde los métodos de la misma clase.
- virtual - es aplicable sólo a los métodos de clase (en ningún caso a los métodos de estructuras), y comunica al compilador que este método tiene que ser colocado en la tabla de las funciones virtuales de la clase.

Clases de memoria

Existen tres clases de memoria: static, input y extern. Estos modificadores de la clase de memoria indican al compilador de una manera explícita que las variables correspondientes se distribuyen en la zona de memoria predeterminada que se llama el pool global. Además, estos modificadores indican un

procesamiento específico de los datos de variables.

Si una variable declarada a nivel local no es estática, entonces la distribución de la memoria para esta variable se realiza de una manera automática en la pila (stack) de programa. La liberación de memoria destinada para un array no estático también se realiza de forma automática cuando se sale fuera de los límites de visibilidad del bloque, en el cual este array ha sido declarado.

Véase también

Tipos de datos, Encapsulación y extensión de tipos, Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos, Miembros estáticos de la clase

Variables locales

Una variable declarada dentro de una función función es local. La zona de visibilidad de la variable local está limitado por los márgenes de la función dentro de la cual ésta esta declarada. La variable local puede ser inicializada con la ayuda de cualquier expresión. La inicialización de la variable local se realiza cada vez durante la llamada a la función correspondiente. Las variables locales se ubican en zona temporal de la memoria de las función correspondiente.

Ejemplo:

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

La zona de aplicación(o visibilidad) de una variable es una parte del programa en la cual se puede referirse a la variable. Las variables declaradas dentro del bloque (a nivel local) van a tener el bloque como su zona de aplicación. La zona de aplicación del bloque se empieza con la declaración de la variable y se termina con la llave derecha final.

Las variables locales declaradas al principio de la función también tienen la zona de aplicación del bloque igual que los parámetros de la función, que también son variables locales. Cualquier bloque puede contener las declaraciones de las variables. Si los bloques están anidados y el nombre de Identificador del bloque exterior coincide con el del bloque interior, el identificador del bloque exterior está "invisible" (oculto) hasta que se termine el trabajo en el bloque interior.

Ejemplo:

```
void OnStart()
  {
//---
   int i=5;       // variable local de la función
     {
      int i=10;  // variable de la función
      Print("En el bloque i = ",i);    // resultado  i = 10;
     }
   Print("Fuera del bloque i = ",i);  // resultado  i = 5;
  }
```

Eso significa que mientras se ejecuta el bloque interno, él ve los valores de sus propios identificadores locales, y no los valores de los identificadores con los mismos nombres en el bloque exterior.

Ejemplo:

```
void OnStart()
  {
//---
   int i=5;       // variable local de la función
   for(int i=0;i<3;i++)
      Print("Dentro for i =",i);
   Print("Fuera del bloque i =",i);
```

```
  }
/* Resultado de ejecución
Dentro for i = 0
Dentro for i = 1
Dentro for i = 2
Fuera del bloque i = 5
*/
```

Las variables locales declaradas como static, tienen la visibilidad del bloque, a pesar de que existan desde el principio de ejecución del programa.

Pila

En cada programa MQL5 para guardar las funciones variables locales que se crean automáticamente se asigna una zona de memoria especial que se llama la pila. Una pila se asigna para todas las funciones, y por defecto su tamaño es de 256 kb. Usted puede cambiar el tamaño de la pila usando la directiva del compilador #property stacksize.

Las variables locales estáticas se ubican en el mismo lugar que las demás variables estáticas y globales, en una zona especial de la memoria que existe separadamente de la pila. Las variables creadas dinámicamente también utilizan una zona de memoria distinta de la pila.

Con cada llamada a la función, para las variables internas no estáticas se les asigna una zona en la pila. Después de salir de la función, la memoria se queda disponible para su volver a usarse.

Si desde la primera función se hace la llamada a la segunda, ésta en su lugar ocupa en la pila un volumen necesario para sus variables en la memoria restante de la pila. De esta manera, cuando se utilizan las funciones incluidas, la memoria de la pila será ocupada consecutivamente por cada función. Esto puede provocar la falta de memoria durante la siguiente llamada a la función. Esta situación se llama sobrellenado de memoria.

Por eso es mejor utilizar la memoria dinámica para los grandes datos locales - cuando se entra en la función, asignar la memoria necesaria para uso local en el sistema (new, ArrayResize()), y cuando se sale de la función, liberar la memoria (delete, ArrayFree()).

Véase también

Tipos de datos, Encapsulación y extensión de tipos, Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Parámetros formales

Los parámetros pasados a la función son locales. Su zona de visibilidad es el bloque de la función. Los parámetros formales tienen que tener nombres distintos que las variables externas y variables locales definidas dentro de la función. En el bloque de la función a los parámetros formales se les puede asignar algunos valores. Si el parámetro formal está declarado con el modificador const, su valor no se puede cambiar dentro de la función.

Ejemplo:

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

Los parámetros formales pueden ser inicializados por las constantes. En este caso, el valor inicializado se considera como el valor por defecto. Los parámetros que siguen después del parámetros inicializado también tienen que ser inicializado.

Ejemplo:

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

Cuando se llama una función como esa, se puede omitir los parámetros inicializados, a su vez serán puestos los valores por defecto.

Ejemplo:

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

Los parámetros de tipos simples se traspasan por valor, es decir, los cambios de la correspondiente variable local de este tipo dentro de la función llamada no se reflejará de ninguna manera en la función llamadora. Los arrays de cualquier tipo y los datos del tipo de estructuras siempre se pasan por referencia. Si es preciso prohibir el cambio del array o el contenido de la estructura, los parámetros de estos tipos tienen que declararse con la palabra clave const.

También existe la posibilidad de pasar por referencia los parámetros de tipos simples. En este caso, la modificación de estos parámetros se reflejará en las correspondientes variables en la función llamada, que han sido pasadas por referencia. Para especificar que un parámetro se pasa por referencia, después del tipo de datos hay que poner el modificador &.

Ejemplo:

```
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;
  }
```

No se puede inicializar los parámetros pasados por referencia con valores por defecto.

No se puede pasar a la función más de 64 parámetros.

Véase también

Variables Input, Tipos de datos, Encapsulación y extensión de tipos, Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Variables estáticas

La clase de memoria static define una variable estática. El modificador static se indica antes del tipo de datos.

Ejemplo:

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

Una variable estática puede ser inicializada por una constante correspondiente a su tipo o por una expresión constante, a diferencia de una variable local simple, que puede ser inicializada por cualquier expresión.

Las variables estáticas existen a partir del momento de ejecución del programa y son inicializadas sólo una vez antes de llamar a la función especializada OnInit(). Si los valores iniciales no están especificados, entonces las variables de la clase estática de la memoria adquieren los valores iniciales cero.

Las variables locales declaradas con la palabra clave static conservan sus valores durante todo el tiempo de vida de la función. Con cada siguiente llamada a la función, estas variables locales ya contienen los valores que tenían durante la llamada anterior.

Cualquier variable en el bloque, salvo los parámetros formales de una función, puede ser definida como estática.

Ejemplo:

```
int Counter()
  {
   static int count;
   count++;
   if(count%100==0) Print("La función Counter ha sido llamada ya ",count," veces");
   return count;
  }
void OnStart()
  {
//---
   int c=345;
   for(int i=0;i<1000;i++)
     {
      int c=Counter();
     }
   Print("c = ",c);
  }
```

Véase también

Tipos de datos, Encapsulación y extensión de tipos, Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos, Miembros estáticos de la clase

Variables globales

Las variables globales se creen mediante la colocación de sus declaraciones fuera de la descripción de una función. Las variables globales se definen al mismo nivel que las funciones, es decir, no son locales en ningún bloque.

Ejemplo:

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

La visibilidad de las variables globales es el programa entero, y están disponibles desde todas las funciones definidas en el programa. Éstas se inicializan por cero, si no está definido explícitamente otro valor inicial. Una variable global puede ser inicializada sólo por una constante correspondiente a su tipo o por una expresión constante.

La inicialización de las variables globales se realiza sólo una vez justo después de cargarse el programa en la memoria del terminal de cliente.

Nota: no se puede confundir las variables declaradas a nivel global con las variables globales del terminal de cliente, el acceso a las cuales se consigue a través de la función GlobalVariable...().

Véase también

Tipos de datos, Encapsulación y extensión de tipos, Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

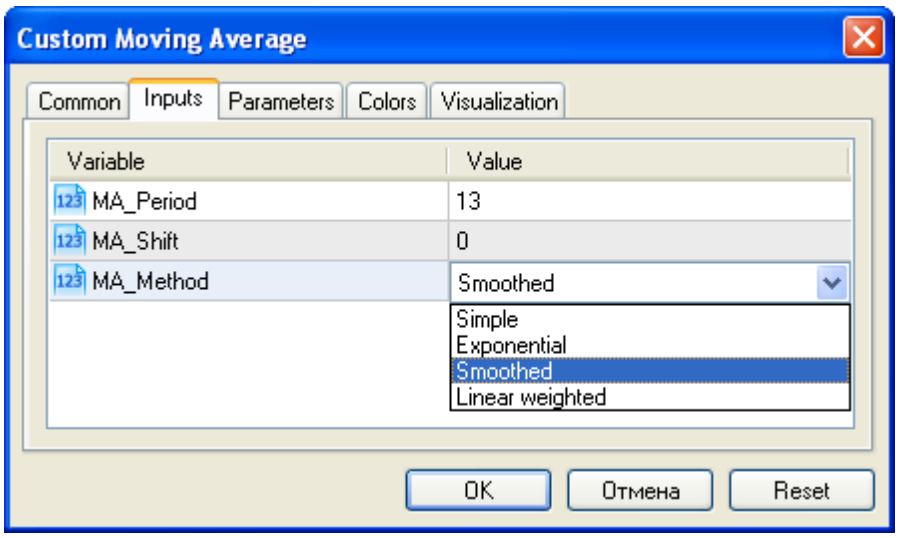
Variables Input

La clase de memoria input define una variable externa. El modificador input se indica antes del tipo de datos. No se puede cambiar el valor de una variable con el modificador input dentro del programa mql5, estas variables son exclusivamente para la lectura. Sólo el usuario puede modificar los valores de las variables input desde la ventana de propiedades de su programa.

Ejemplo:

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

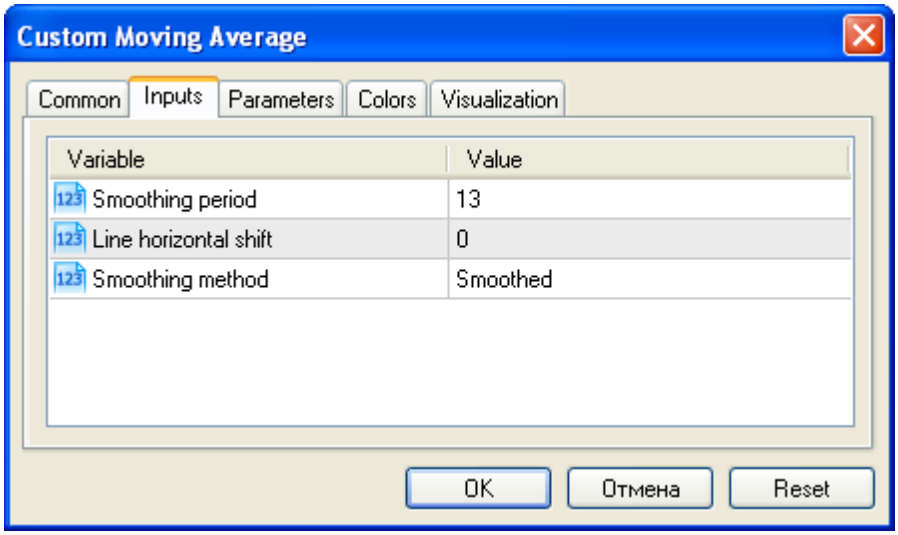
Las variables Input determinan los parámetros de entrada del programa, están disponibles desde la ventana Propiedades de la aplicación.



Es posible establecer otro modo de mostrar los nombres de parámetros de entrada en la pestaña "Inputs". Para eso se utiliza un comentario de cadena, el que tiene que ir después de la descripción del parámetro de entrada en la misma cadena. De esa manera, se puede asignar a los parámetros de entrada los nombres más comprensibles para el usuario.

Ejemplo:

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



Nota: Los arrays y variables de tipos compuestos no pueden actuar como las variables input.

Traspaso de parámetros al llamar a los indicadores personalizados desde los programas mql5

Los indicadores personalizados se invocan con ayuda de la función iCustom(). Allí, después del nombre del indicador personalizado, deben ir los parámetros de acuerdo estricto con la declaración de variables input de este indicador de usuario. Si se indican menos parámetros que las variables input declaradas en el indicador personalizado invocado, entonces los parámetros que faltan se llenan con los valores especificados durante la declaración de variables.

Si en indicador personalizado se usa la función OnCalculate del primer tipo (es decir, el indicador se calcula en el mismo array de datos), entonces uno de los valores ENUM_APPLIED_PRICE o manejador (handle) del otro indicador debe actuar como el último parámetro durante la llamada de este indicador de usuario. Entonces, todos los parámetros correspondientes a las variables input tienen que ser claramente indicados.

Enumeraciones como parámetro input

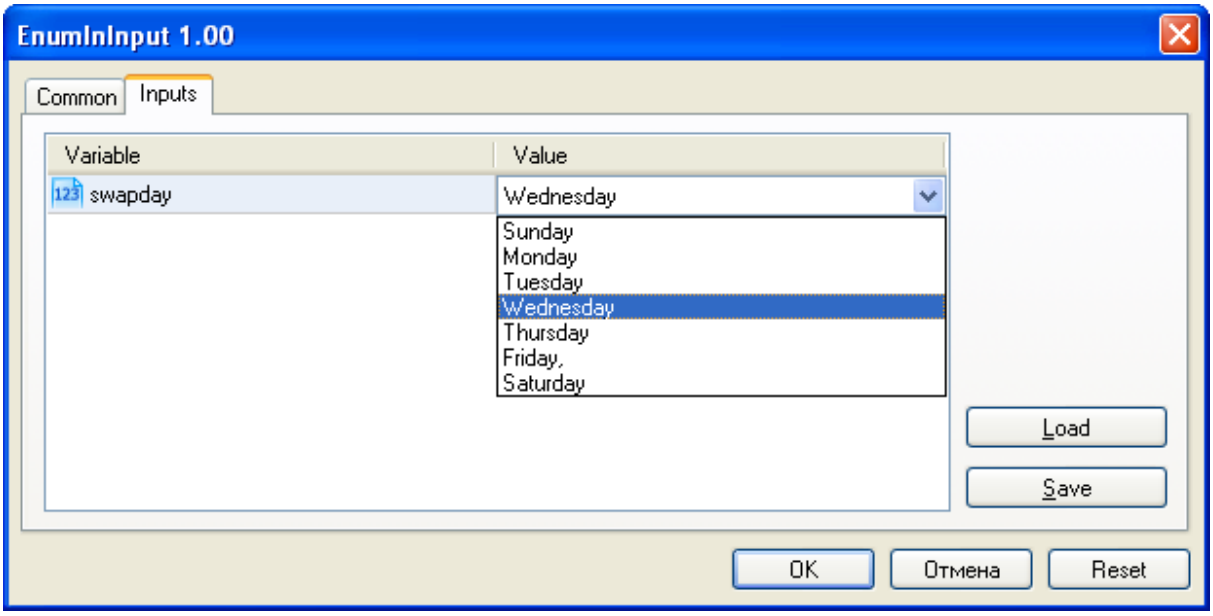
No sólo las enumeraciones "built-in", previstas en el lenguaje MQL5, pueden ser utilizadas como las variables input (parámetros de entrada para los programas mql5), sino las enumeraciones definidas por el usuario. Por ejemplo, podemos crear la enumeración dayOfWeek que describe los días de la semana, y usar la variable input para indicar un día concreto de la semana no como un número, sino en forma más común para el usuario.

Ejemplo:

```
#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;
```

Para que el usuario pueda eligir el valor necesario de la ventana de Propiedades durante el inicio del script, usamos el comando de preprocesador #property script_show_inputs. Iniciamos el script y podemos eligir de la lista uno de los valores de la enumeración dayOfWeek. Iniciamos el script EnumInInput y pasamos a la pestaña "Parámetros. Por defecto, el valor del parámetro swapday (día de cómputo del triple swap) es miércoles (W=3), pero nosotros podemos especificar cualquier otro valor y utilizar este valor para cambiar la operación del programa.



La cantidad de posibles valores de enumeración está limitada. Por eso para eligir un valor de entrada se utiliza una lista desplegable. Los nombres mnemotécnicos de los elementos de enumeración se usan en calidad de los valores mostrados dentro de la lista. Pero si un nombre mnemotécnico está asociado con un comentario, como se muestra en el ejemplo, entonces en vez del nombre mnemotécnico se usa el contenido del comentario.

Cada valor de la enumeración dayOfWeek tiene su propio valor de 0 a 6, pero en la lista de parámetros se mostrarán los comentarios indicados para cada valor. Eso da una flexibilidad adicional para escribir los programas con las descripciones claras de los parámetros de entrada.

Cada valor de la enumeración dayOfWeek tiene su valor de 0 a 6, pero en la lista de parámetros serán mostrados los cometarios especificados para cada valor. Esto oporta la flexibilidad adicional para escribir un programa con las descripciones claras de los parámetros de entrada.

Variables con modificador sinput

Las variables con modificador input no sólo permiten establecer los valores de parámetros externos al

iniciar los programas, sino también juegan un papel muy importante durante la optimización de estrategias comerciales en el probador. Cada variable input declarada en el EA, salvo el tipo string, puede participar en la optimización.

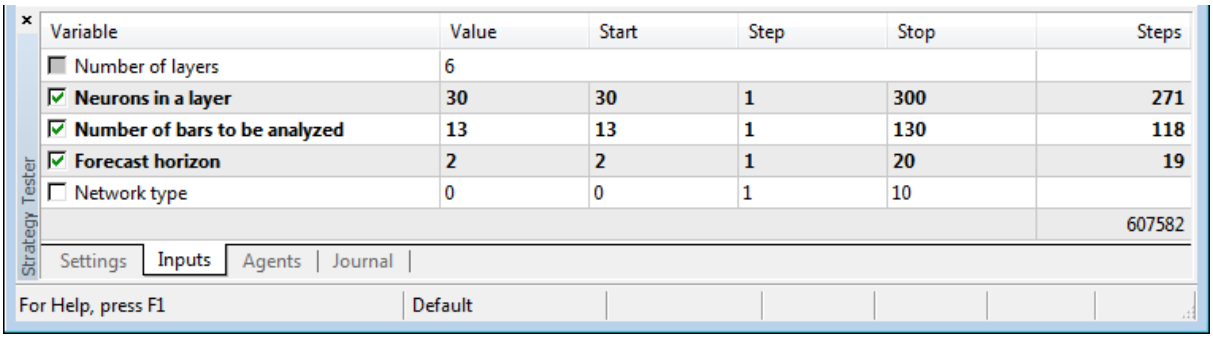
En algunas ocasiones resulta necesario excluir algunos parámetros externos del programa de la formación de las áreas de posibles pasos en el probador. Precisamente para estas ocasiones existe el modificador de memoria sinput. sinput es la sigla de declaración de una variable estática externa: sinput = static input. Es decir, esta declaración en el código del EA

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

equivale a la declaración completa

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

Una variable declarada con el modificador sinput es un parámetro de entrada del programa MQL5, y el valor de este parámetro se puede cambiar durante el arranque. Pero esta variable no participa en el proceso de optimización de los parámetros de entrada, es decir, no se efectúa el repaso de sus valores durante la búsqueda del mejor conjunto de parámetros según el criterio establecido.



En la imagen se muestra que el EA tiene 5 parámetros externos, de los cuales el parámetro "Número de capas" ha sido declarado como sinput y es igual a 6. Este parámetro no puede cambiarse durante el proceso de optimización de la estrategia de trading. Sólo se puede establecer para él un valor necesario que va a utilizarse. Los campos Inicio, Paso y Stop para esta variable no están disponibles para poner sus valores.

De esta manera, si establecemos el modificador sinput para una variable, el usuario tiene prohibido optimizar este parámetro. Eso significa que en el Probador de Estrategias el usuario no podrá establecer el valor inicial y final para esta variable con el fin del repaso automático dentro del rango establecido durante el proceso de optimización.

Pero hay una excepción en esta regla: las variables sinput se puede variar en las tareas de optimización utilizando la función ParameterSetRange(). Esta función ha sido creada especialmente para el control programado del área de valores disponibles para cualquier variable input, incluyendo las que han sido declaradas como static input (sinput). Otra función ParameterGetInput() permite obtener los valores de las variables input cuando se inicia la optimización (en el manejador OnTesterInit()), y en caso de necesidad redefinir el paso de cambio y el rango dentro del cual va a repasarse el valor del parámetro a optimizar.

De esta manera, la combinación del modificador sinput con otras dos funciones para el trabajo con los parámetros input permite crear unas reglas sumamente flexibles para establecer los intervalos de optimización de unas variables input dependiendo del valor de otras variables input.

Véase también

iCustom, Enumeraciones, Propiedades de programas

Variables Extern

La palabra clave extern se utiliza para declarar los identificadores de las variables como los identificadores de la clase estática de memoria con el tiempo de vida global. Estas variables existen desde el momento del inicio de programa, y la memoria para ellas se asigna y se inicializa inmediatamente después del inicio del programa.

Se puede crear programas compuestos de varios archivos iniciales. En este caso, para el preprocesador se utiliza el comando #include. Las variables declaradas como extern con el mismo tipo e identificador, pueden existir en diferentes archivos iniciales de un proyecto.

Durante la compilación de todo el proyecto, todas las variables extern con el mismo tipo e identificador, se asocian con una parte de memoria del pool de las variables globales. Las variables extern son útiles para una compilación separada de los archivos iniciales. Las variables extern pueden ser inicializadas, aunque sólo una vez. Está prohibida la existencia de varias variables inicializadas extern del mismo tipo y con el mismo identificador.

Véase también

Tipos de datos, Encapsulación y extensión de tipos, Inicialización de variables, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Inicialización de variables

Cualquier variable puede ser inicializada durante la definición. Si la variable no es inicializada explícitamente, el valor almacenado en esta variable puede ser cualesquiera. La inicialización implícita no se realiza.

Las variables globales y estáticas pueden ser inicializadas sólo por la constante del tipo correspondiente o por una expresión constante. Las variables locales pueden ser inicializadas por cualquier expresión, y no sólo por una constante.

La inicialización de las variables globales y estáticas se realiza sólo una vez. La inicialización de las variables locales se realiza cada vez al llamar a la función correspondiente.

Ejemplos:

```
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 } };
//--- de tetris
int    right[4]={WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER,
                 WIDTH_IN_PIXELS+VERT_BORDER,WIDTH_IN_PIXELS+VERT_BORDER};
//--- inicialización de todos los campos de la estructura con valores cero
MqlTradeRequest request={0};
```

La lista de valores de los elementos del array tiene que encerrarse dentro de las llaves. Las sucesiones inicializadoras perdidas se consideran iguales a 0. En la secuencia que inicializa tiene que haber por lo menos un valor: este valor inicializa el primer elemento de la estructura correspondiente o del array, los elementos que faltan se consideran iguales a cero.

Si el tamaño del array inicializado no está especificado, éste es determinado por el compilador basándose en el tamaño de la sucesión inicializadora. Los arrays multidimensionales no pueden ser inicializados por una sucesión unidimensional (por una sucesión sin llaves adicionales), excepto el caso, cuando se especifica sólo un elemento inicializador (cero, por regla general).

Los arrays (incluso aquellos que hayan sido declarados a nivel local) pueden ser inicializados sólo por las constantes.

Ejemplos:

```
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]);
  }
```

Para las variables del tipo de estructuras se permite una inicialización parcial, lo mismo se refiere a los arrays estáticos (con un tamaño fijado explícitamente). Se puede inicializar uno o varios primeros elementos de la estructura o array, el resto de los elementos en este caso serán inicializados con ceros.

Véase también

Tipos de datos, Encapsulación y extensión de tipos, Visibilidad y tiempo de vida de variables, Creación y eliminación de objetos

Visibilidad y tiempo de vida de variables

Hay dos tipos principales de zona de visibilidad: zona local de visibilidad y zona global de visibilidad.

Una variable declarada fuera de todas las funciones se coloca en la visibilidad global. Estas variables pueden ser accesibles desde cualquier parte del programa. Estas variables se ubican en el pool global de la memoria, por eso el tiempo de sus vidas coincide con el tiempo de vida del programa.

Una variable declarada dentro del bloque (parte del código encerrada dentro de las llaves) pertenece a la visibilidad local. Esta variable no es visible (por tanto no es accesible) fuera del bloque en el que está declarada. El caso más común de la declaración local es una variable declarada dentro de la función. Una variable con declaración local se coloca en una pila, y su tiempo de vida coincide con el tiempo de vida de la función.

Ya que la zona de visibilidad de una variable local es el bloque donde está declarada, hay posibilidad de declarar las variables con los nombres que coincidan con los de las variables declaradas en otros bloques, y también declaradas en los niveles superiores, así sucesivamente hasta el nivel global.

Ejemplo:

```
void CalculateLWMA(int rates_total,int prev_calculated,int begin,const double &price[
  {
   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;
     }
//---
  }
```

Preste la atención a la variable i declarada en línea

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

Su visibilidad es sólo el ciclo for, fuera de este ciclo hay otra variable con el mismo nombre que ha sido declarada al principio de la función. Además, en el cuerpo del ciclo está declarada la variable K, cuya zona de visibilidad es el cuerpo del ciclo.

Se puede declarar las variables locales con el especificador de acceso static. En este caso, el compilador coloca esta variable en el pool global de la memoria. Por tanto, el tiempo de vida de una variable estática coincide con el tiempo de vida del programa. No obstante, la zona de visibilidad de esta variable se limita por el bloque donde está declarada.

Véase también

Tipos de datos, Encapsulación y extensión de tipos, Inicialización de variables, Creación y eliminación de objetos

Creación y eliminación de objetos

Una vez cargado el programa mql5 para ser ejecutado, a cada constante se le asigna una parte de memoria de acuerdo con el tipo de la variable. Dependiendo del nivel de acceso las variables se dividen en dos tipos: variables globales y variables locales, y también dependiendo de las clases de memoria: parámetros input del programa mql5, estáticas y automáticas. En caso de necesidad cada variable es inicializada por el valor correspondiente. Después de ser usada, la variable se deinicializa, y la memoria ocupada por ésta se devuelve al sistema ejecutivo MQL5.

Inicialización y deinicialización de variables globales

Las variables globales se inicializan de una forma automática inmediatamente después de que se cargue el programa mql5 y antes de que se llame a cualquier función. Durante la inicialización se asignan valores iniciales a las variables de tipos simples y se llama al constructor (si existe) para los objetos. Los parámetros input siempre se declaran a nivel global, son inicializados por valores asignados por los usuarios en un diálogo durante el inicio del programa mql5.

A pesar de que las variables estáticas suelen declararse a nivel local, la memoria para estas variables se distribuye de antemano, y la inicialización se realiza justo después de que el programa se cargue, lo mismo que pasa con las variables globales.

El orden de inicialización corresponde al orden de declaración de una variable en el programa, y la deinicialización se realiza a la inversa antes de cargar el programa. Esta regla vale sólo para la variables que no han sido creadas por el operador new. Estas variables se crean y se inicializan automáticamente justo después de la carga, y se deinicializan directamente antes de la descarga del programa.

Inicialización y deinicialización de variables locales

Si una variable declarada a nivel local no es estática, entonces la distribución de memoria para esta variable se realiza de una forma automática. Las variables locales, igual como globales, se inicializan automáticamente en el momento cuando la ejecución del programa encuentra la declaración de una variable local. De esta manera, el orden de inicialización corresponde al orden de declaración.

Las variables locales se deinicializan al final del bloque del programa en el cual ha sido declaradas, y en el orden inverso a la declaración. Un bloque del programa es un operador compuesto que puede ser una parte del operador de elección switch, ciclo (for, while, do-while), cuerpo de función o parte del operador if-else.

Las variables locales se inicializan sólo en el momento cuando la ejecución del programa llega a la declaración de variable. Si durante el proceso de ejecución del programa el bloque en el que una variable está declarada, no ha sido ejecutado, esta variable no se inicializa.

Inicialización y deinicialización de objetos colocados dinámicamente

Un caso especial representan los punteros a objetos, puesto que la declaración de un puntero no implica la posterior inicialización del objeto correspondiente. Los objetos colocados dinámicamente se inicializan sólo en el momento cuando la muestra de clase es creada por el operador new. La inicialización del objeto supone la invocación del constructor de la clase correspondiente. Si en la clase

no hay constructores correspondientes, entonces sus elementos que tienen el tipo simple, no serán inicializados automáticamente; los elementos de tipos string, array dinámico y objeto compuesto serán inicializados de una forma automática.

Los punteros pueden ser declarados a nivel local o global y pueden ser inicializados por el valor vacío NULL o por el valor del puntero del mismo tipo o del tipo derivado. Si el operador *new* ha sido invocado para un puntero declarado a nivel local, entonces el operador *delete* para este puntero tiene que ser ejecutado antes de salir de este nivel. En caso contrario, el puntero se perderá y el objeto no podrá ser eliminado de una forma explícita.

Todos los objetos creados por la expresión *puntero_objeto*=new *Nombre_Clase*, a continuación tienen que ser eliminados obligatoriamente por el operador delete(*puntero_objeto*). Si por alguna razón esta variable no ha sido eliminada por el operador delete después de que el programa complete su trabajo, aparecerá un mensaje de entrada en el apartado "Expertos". Es posible declarar varias variables, y asignarles a todas el puntero a un objeto.

Si el objeto creado dinámicamente tiene un constructor, este constructor será llamado en el momento de la ejecución del operador *new*. Si el objeto tiene un destructor, este destructor será llamado en el momento de la ejecución del operador *delete*.

De esta manera, los objetos colocados dinámicamente se crean sólo en el momento cuando son creados por el operador *new*, y se eliminan de una forma garantizada por el operador *delete* o automáticamente por el sistema ejecutora de MQL5 durante la descarga del programa. El orden de declaración de punteros de los objetos creados dinámicamente no influye en el orden de su inicialización. El programador controla completamente el orden de inicialización y deinicialización.

Características breves de variables

La información general sobre el orden de creación, eliminación, llamada a los constructores y destructores se muestra en la tabla de abajo.

	Variable global automática	Variable local automática	Objeto creado dinámicamente
Inicialización	inmediatamente después de que se cargue el programa mql5	al llegar durante la ejecución a la línea de código, donde ésta está declarada	durante la ejecución del operador **new**
Orden de inicialización	en el orden de declaración	en el orden de declaración	no depende del orden de declaración
Deinicialización	antes de que se descargue el programa mql5	cuando la ejecución abandona el bloque de declaración	durante la ejecución del operador **delete** o antes de que se descargue el programa mql5
Orden de deinicialización	en el orden inverso a la inicialización	en el orden inverso a la inicialización	no depende del orden de inicialización
Llamada al constructor	al cargar el programa mql5	durante la inicialización	durante la ejecución del operador *new*

Llamada al destructor	al descargar el programa mql5	al salir del bloque donde la variable ha sido inicializada	durante la ejecución del operador *delete*
Aviso de errores	aviso en el apartado "Expertos" sobre el intento de eliminación del objeto creado automáticamente	aviso en el apartado "Expertos" sobre el intento de eliminación del objeto creado automáticamente	aviso en el apartado "Expertos" sobre los objetos creados dinámicamente pero con fallos durante la descarga del programa mql5

Véase también

Tipos de datos, Encapsulación y extensión de tipos, Inicialización de variables, Visibilidad y tiempo de vida de variables

Preprocesador

El preprocesador es un subsistema especial del compilador MQL5 que se ocupa de la preparación preliminar del código fuente del programa directamente antes de su compilación.

El preprocesador permite mejorar la legibilidad del código fuente. Se puede conseguir la estructurización del código mediante la inclusión de ciertos archivos con los códigos fuentes de los programas mql5. Además de eso, la posibilidad de asignar los nombres mnemotécnicos a algunas constantes también contribuye a la mejora de legibilidad del código.

El preprocesador, asimismo, permite determinar los parámetros especiales de los programas mql5:

- Declarar constantes
- Establecer propiedades de programa
- Incluir archivos en texto de programa
- Importar funciones

Si el signo # se usa como el primer símbolo en la línea del programa, entonces esta línea es directiva del preprocesador. Una directiva del preprocesador se termina con el carácter de nueva línea.

Declaración de constante (#define)

La directiva #define puede ser utilizada para la asignación de los nombres mnemotécnicos a las constantes. Hay dos formas:

```
#define identifier expression                    // forma libre de parámetros
#define identifier(par1,... par8) expression     // forma paramétrica
```

La directiva #define sustituye todas las siguientes entradas identifier en el texto original por **expression**. identifier se sustituye sólo en el caso si es un token suelto. identifier no se sustituye si es parte del un comentario, una cadena o parte de otro identificador más largo.

El identificador de constante se rige por las mismas reglas, las que funcionan para los nombres de las variables. El valor puede ser de cualquier tipo:

```
#define ABC               100
#define PI                3.14
#define COMPANY_NAME      "MetaQuotes Software Corp."
...
void ShowCopyright()
  {
   Print("Copyright  2001-2009, ",COMPANY_NAME);
   Print("http://www.metaquotes.net");
  }
```

expression puede estar compuesto por varios tokens, como por ejemplo, palabras claves, constantes, expresiones constantes y no constantes. **expression** se termina con el fin de la línea y no puede saltar a la siguiente.

Ejemplo

```
#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);
  }
/* Resultado
   2 + 3*2 = 8
   (2 + 3)*2 = 10
*/
```

Forma paramétrica #define

En caso de forma paramétrica, todas las siguientes entradas encontradas del identifier serán reemplazadas por expression, tomando en consideración los parámetros actuales. Por ejemplo,

```
 // un ejemplo con dos parámetros a y b
#define A 2+3
#define B 5-1
#define MUL(a, b) ((a)*(b))

double c=MUL(A,B);
Print("c=",c);
/*
expresión double c=MUL(A,B);
es equivalente a double c=((2+3)*(5-1));
*/
// Resultado
// c=20
```

Asegúrese de encerrar los parámetros entre paréntesis a la hora de usarlos en expression, porque esto va a permitirle evitar los errores implícitos, difíciles de detectar. Si reescribimos el ejemplo sin utilizar los paréntesis, el resultado será totalmente diferente:

```
 // un ejemplo con dos parámetros a y b
#define A 2+3
#define B 5-1
#define MUL(a, b) a*b

double c=MUL(A,B);
Print("c=",c);
/*
expresión double c=MUL(A,B);
es equivalente a double c=2+3*5-1;
*/
// Resultado
// c=16
```

Cuando se utiliza la forma paramétrica, se admite el máximo de 8 parámetros.

```
 // forma paramétrica correcta
#define LOG(text)  Print(__FILE__,"(",__LINE__,") :",text)   // un parámetro - 'text'

 // forma paramétrica incorrecta
#define WRONG_DEF(p1, p2, p3, p4, p5, p6, p7, p8, p9)   p1+p2+p3+p4 // más de 8 parám
```

Véase también

Identificadores, Constantes de caracteres

Propiedades de programas (#property)

Para cada programa mql5 se puede indicar unos parámetros específicos adicionales *#property*, los que ayudan al terminal de cliente prestar un servicio correcto a los programas sin necesidad de iniciarlos explícitamente. En primer lugar, esto se refiere a los ajustes de los indicadores externos. Las propiedades descritas en en los archivos incluidos se ignoran por completo. Las propiedades tienen que ser especificadas en el archivo-mql5 principal.

```
#property identificador valor
```

El compilador apuntará los valores renovados en los ajustes del módulo ejecutado.

Constante	Tipo	Descripción
icon	string	ruta del archivo con la imagen que va a mostrarse como el icono del programa EX5. Las reglas para especificar la ruta son las mismas que para los recursos. La propiedad tiene que especificarse en el módulo principal con el código fuente MQL5. El archivo del icono debe ser del formato ICO.
link	string	referencia a la web de la empresa productora
copyright	string	nombre de la empresa productora
version	string	versión del programa, no más de 31 caracteres
description	string	breve descripción textual del programa mql5. Puede haber algunas description, cada una de las cuales describe una línea de texto. La longitud total de todas las description no puede superar más de 511 caracteres incluyendo saltos de líneas
stacksize	int	tamaño de pila para las llamadas recursivas
library		biblioteca; no asigna ninguna función de inicio; funciones con modificador export se puede importar en otros programas mql5

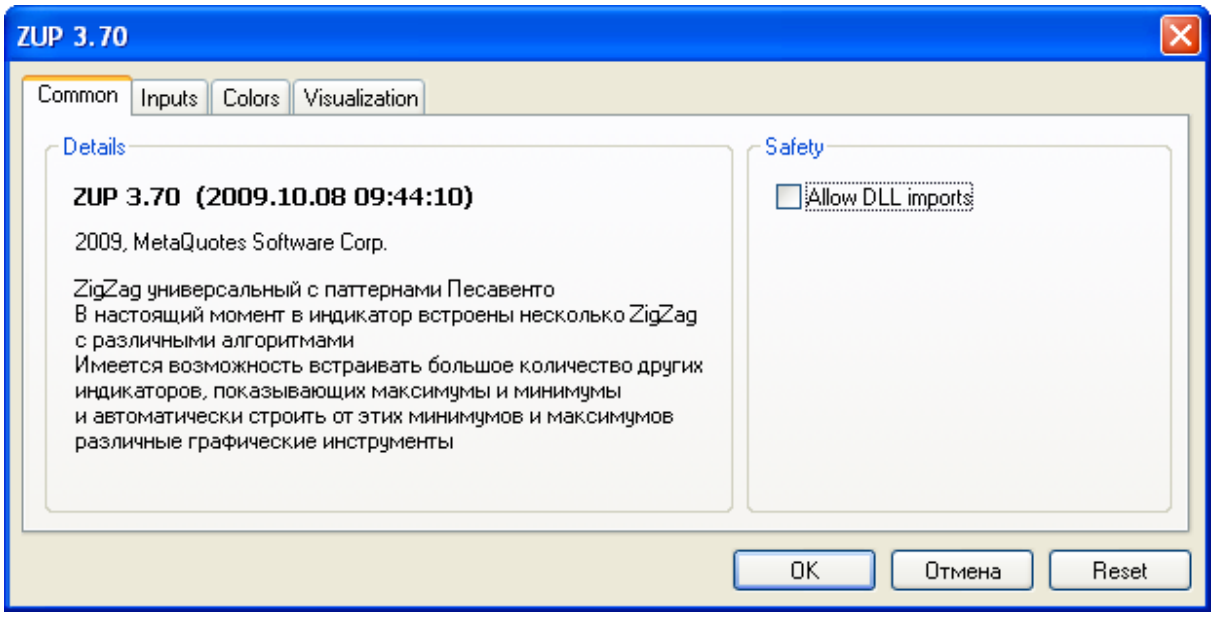
indicator_applied_price	int	especifica el valor por defecto para el campo "Apply to". Se puede especificar uno de los valores de enumeración ENUM_APPLIED_PRICE. Si la propiedad no está especificada, entonces por defecto se aplica el valor PRICE_CLOSE
indicator_chart_window		mostrar el indicador en la ventana del gráfico
indicator_separate_window		mostrar el indicador en la ventana separada
indicator_height	int	el alto fijo de la subventana del indicador en píxeles (propiedad INDICATOR_HEIGHT)
indicator_buffers	int	cantidad de buffers para el cálculo del indicador
indicator_plots	int	cantidad de series gráficas en el indicador
indicator_minimum	double	margen inferior de escala de la ventana separada del indicador
indicator_maximum	double	margen superior de escala de la ventana separada del indicador
indicator_labelN	string	pone una marca para la enésima serie gráfica mostrada en la ventana DataWindow. Para las series gráficas que necesitan varios buffers de indicadores (DRAW_CANDLES, DRAW_FILLING y los demás), los nombres de las marcas vienen con separador ';'.
indicator_colorN	color	color para mostrar la línea N, donde la N es el número de la serie gráfica; la enumeración se empieza desde 1
indicator_widthN	int	grosor de línea en la serie gráfica, donde la N es el número de la serie gráfica; la

		enumeración se empieza desde 1
indicator_styleN	int	estilo de línea la serie gráfica especificado con el valor de ENUM_LINE_STYLE. La N es el número de la serie gráfica, la enumeración se empieza desde 1
indicator_typeN	int	tipo de construcción gráfica especificado con el valor de ENUM_DRAW_TYPE. La N es el número de la serie gráfica, la enumeración se empieza desde 1
indicator_levelN	double	nivel horizontal N en la ventana separada del indicador
indicator_levelcolor	color	color de niveles horizontales del indicador
indicator_levelwidth	int	grosor de niveles horizontales del indicador
indicator_levelstyle	int	estilo de niveles horizontales del indicador
script_show_confirm		mostrar ventana de confirmación antes de iniciar un script
script_show_inputs		mostrar ventana con las propiedades antes de iniciar un script y prohibir la muestra de la ventana de confirmación
tester_indicator	string	Nombre del indicador personalizado en el formato "*nombre_de_indicador.ex5*". Los indicadores que requieren verificaciones se definen automáticamente desde la llamada a la función iCustom (), si el parámetro correspondiente es fijado por una cadena constante. Para los demás casos (uso de la función IndicatorCreate() o uso de una cadena no constante en el parámetro que asigna el nombre del

		indicador) se necesita esta propiedad
tester_file	string	Nombre del archivo incluyendo su extensión, que se pasa al comprobador para que éste lo procese, tiene que ir encerrado entre comillas dobles (como una cadena constante). Siempre hay que especificar los archivos de entrada, en caso de que sean necesarios
tester_library	string	Nombre de la biblioteca con su extensión encerrado entre comillas dobles. Una biblioteca puede tener tanto la extensión dll como la ex5. Las bibliotecas que necesitan verificación se determinan automáticamente. Sin embargo, si una de las bibliotecas se usa por un indicador personalizado, hace falta usar esta propiedad

Ejemplo de descripción y número de una versión

```
#property version     "3.70"      // versión actual de Asesor Experto
#property description "ZigZag universal con patrones de Pesavento"
#property description "Actualmente en el indicador están integrados varios ZigZag con
#property description "Existe posibilidad de integrar una cantidad importante de otro
#property description "mínimos, y construir automáticamente a base de éstos diferente
```



Ejemplo de especificar una marca separada para cada búfer de indicadores ("C open;C high;C low;C close")

```
#property indicator_chart_window
#property indicator_buffers 4
#property indicator_plots   1
#property indicator_type1   DRAW_CANDLES
#property indicator_width1  3
#property indicator_label1  "C open;C high;C low;C close"
```

Inclusión de archivos (#include)

La línea de comando *#include* puede encontrarse en cualquier parte del programa, pero normalmente las inclusiones suelen encontrarse al principio del archivo del código fuente. Formato de llamada:

```
#include <nombre_de_archivo>
#include "nombre_de_archivo"
```

Ejemplos:

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

El preprocesador sustituye la cadena *#include <nombre_de_archivo>* por el contenido del archivo WinUser32.mqh. Los paréntesis angulares significan que el archivo WinUser32.mqh va a ser cogido desde un directorio estándar (suele ser *directorio_de_terminal\MQL5\Include*). El directorio corriente no se mira.

Si el nombre del archivo se encierra entre las comillas dobles, entonces la búsqueda se efectua en el directorio corriente (donde se encuentra el archivo principal del código fuente). El directorio estándar no se mira.

Véase también

Biblioteca estándar, Importación de funciones

Importación de funciones (#import)

Las funciones se importan desde los módulos compilados MQL5 (archivos *.ex5) y desde los módulos del sistema operativo (archivos *.dll). El nombre del módulo se indica en la directiva *#import*. Para que el compilador pueda hacer correctamente la llamada a la función importada y organizar el correcto traspaso de parámetros, hace falta la descripción completa de las funciones. La descripción de la función sigue directamente después de la directiva *#import "nombre de módulo"*. El nuevo comando *#import* (puede ir sin parámetros) cierra el bloque de descripción de las funciones importadas.

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

Las funciones importadas pueden tener cualquier nombre. Las funciones con nombres iguales pueden ser importadas a la vez desde diferentes módulos. Las funciones importadas pueden tener nombres que coincidan con los nombres de las funciones built-in. La operación de resolución de contexto determina cuál de las funciones tiene que ser invocada.

La orden de búsqueda del archivo especificado tras la palabra clave #import se describe en la sección Llamadas a las funciones importadas.

Dado que las funciones importadas se encuentran fuera del módulo compilado, el compilador no puede comprobar si los parámetros pasados son correctos. Así que para evitar los errores de ejecución, hay que describir con exactitud la composición y el orden de los parámetros pasados a las funciones importadas. Los parámetros pasados a funciones importadas (tanto de EX5, como de DLL-módulos) pueden tener valores por defecto.

En funciones importadas no se puede usar lo siguiente en calidad de parámetros:

- punteros (*);
- referencias a objetos que contienen arrays dinámicos y/o punteros.

Las clases, arrays de cadenas o objetos complejos que contienen las cadenas y/o arrays dinámicos de cualquier tipo no se puede pasar como parámetros a las funciones importadas de DLL.

Ejemplos:

```
#import "user32.dll"
int    MessageBoxW(uint hWnd,string lpText,string lpCaption,uint uType);
#import "stdlib.ex5"
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
```

Para importar funciones durante la ejecución del programa mql5 se usa la aligación temprana. Eso significa que la biblioteca se carga durante el proceso de carga del programa ex5 que la utilice.

No se recomienda usar el nombre totalmente calificado del módulo cargable del tipo *Drive:\Directory \FileName.Ext*. Las bibliotecas MQL5 se cargan de la carpeta *terminal_dir\MQL5\Libraries*.

Véase también

Inclusión de archivos

Programación orientada a objetos

Programación orientada a objetos (POO) es programación enfocada a los datos, siendo de notar que los datos y el comportamiento están vinculados indisolublemente entre si. Los datos juntamente con el comportamiento constituyen una clase, y los objetos son instancias de clases.

Los componentes de la programación orientada a objetos son los siguientes:

- Encapsulación y extensión de tipos
- Herencia
- Polimorfismo
- Sobrecarga
- Funciones virtuales

POO considera los cálculos como el modelado del comportamiento. Lo que se modela es un objeto representado por las abstracciones computacionales. Supongamos que queremos escribir el juego muy conocido a todos "Tetris", para eso tenemos que aprender a modelar la aparición de una figura aleatoria compuesta por cuatro cuadrados unidos juntos por sus bordes. Además, hace falta regular la velocidad de caída de esta figura, definir las operaciones de rotación y desplazamiento. En una pantalla los movimientos de figuras están limitados con los márgenes del área de juego, este requisito también debe ser modelado. Aparte de eso, las líneas completas deben desaparecer del vaso y hay que llevar la cuenta de puntos obtenidos durante el juego.

De esa manera, un juego tan fácil de comprender requiere la creación de varios modelos: modelo de pieza, modelo de vaso, modelo de movimiento dentro del vaso, etc. Todos estos modelos son unas abstracciones representadas por los cálculos en el ordenador. Para describir estos modelos se utiliza el concepto de *tipo de dato abstracto, TDA* (o tipo de dato compuesto). Hablando en rigor, el modelo de movimiento de una "figura" en el "vaso" no es un tipo de datos, sino un conjunto de operaciones con los datos de tipo "figura" que utilizan limitaciones de datos de tipo "vaso".

Los objetos son variables de clase. Programación orientada a objetos permite crear y usar TDA de una manera fácil. Programación orientada a objetos utiliza el mecanismo de herencia. La ventaja de herencia consiste en el hecho de que se permite la obtención de tipos derivados de los tipos de datos ya definidos por el usuario. Así que, para crear figuras en tetris, para empezar, es más cómodo crear la clase base Shape, a base de la cual se obtienen los tipos derivados de siete posibles figuras de tetris. El comportamiento de las figuras está determinado en la clase base, mientras que en las derivadas se concreta la realización del comportamiento de cada una de las figuras.

En POO los objetos se hacen responsables de su comportamiento. El programador de TDA debe incluir en él un código para describir cualquier comportamiento que normalmente se puede esperar de los objetos correspondientes. El hecho de que el mismo objeto se responsabilice de su comportamiento facilita considerablemente la tarea de programación para cada usuario de este objeto.

Si queremos dibujar una figura en la pantalla, hemos de saber dónde va a estar su centro y cómo dibujarla. Si una figura separada entiende perfectamente como dibujar a si misma, el programador, al usar esta figura, solo debe enviar al objeto el mensaje "dibujar".

El lenguaje MQL5 es parecido al C++, y también dispone del mecanismo de encapsulación para implementación de TDA. Encapsulación reune en sí, por una parte, los detalles internos de implementación de un tipo en particular, y por otra parte, las funciones accesibles desde fuera que puedan influir sobre los objetos de este tipo. Los detalles de implementación pueden ser inaccesibles

para el programa que utilice este tipo.

Un conjunto de conceptos tiene relación con el concepto de POO, incluyendo los siguientes:

- Simulación de acciones del mundo real
- Disponibilidad de tipos de datos definidos por el usuario
- Ocultación de detalles de implementación
- Posibilidad de usar el código varias veces gracias a la herencia
- Interpretación de la llamada a la función durante la ejecución

Algunos de estos conceptos son bastante vagos, algunos son abstractos, otros son de carácter general.

Encapsulación y extensión de tipos

POO es una manera equilibrada de enfocar la escritura del software. Los datos y el comportamiento están empaquetados juntos. Esta encapsulación crea tipos de datos definidos por el usuario, que amplían los tipos de datos del lenguaje y se interactuan. La extensión de tipos es una posibilidad de agregar al lenguaje los tipos de datos definidos por el usuario, los cuales también pueden ser usados de una manera fácil, igual que los tipos básicos.

Un tipo de dato abstracto, por ejemplo una cadena, es una descripción del comportamiento ideal, muy bien conocido. El usuario de la cadena sabe que las operaciones, tales como la concatenación o la impresión, tienen un cierto comportamiento. Las operaciones de concatenación e impresión se llaman métodos.

La implementación concreta de TDA puede tener ciertas restricciones; por ejemplo, las cadenas pueden ser limitadas en su longitud. Estas limitaciones afectan el comportamiento abierto para todos. Al mismo tiempo, los detalles internos o privados de implementación no afectan directamente el modo como el usuario ve el objeto. Por ejemplo, una cadena a menudo se implementa como un array; mientras que la dirección básica interna de los elementos de este array y su nombre no son tan significativos para el usuario.

La encapsulación es la capacidad de ocultar los detalles internos cuando se proporciona un interfaz abierto al tipo definido por el usuario. En MQL5, igual que en C++, para la provisión de encapsulación se utilizan las definiciones de clase y estructura (class y struct) en combinación con las palabras claves de acceso private (privado), protected (protegido) y public (público).

La palabra clave public indica que el acceso a los elementos que la siguen está abierto sin ningunas restricciones. Sin esta palabra clave los elementos de la clase están cerrados por defecto. Los elementos cerrados están disponibles sólo para las funciones miembro de su clase.

Los elementos de clase protegidos están disponibles para las funciones miembro no sólo de su clase, sino también de las clases heredadas. Los elementos de acceso abiertos están disponibles para cualquier función dentro de la zona de visibilidad de declaración de la clase. La protección permite ocultar una parte de implementación de clase, evitando de esa manera los cambios imprevistos de la estructura de datos. La restricción de acceso o ocultación de datos son particularidades de la programación orientada a objetos.

Habitualmente procuran proteger los elementos de clase y declararlos con el modificador protected. El establecimiento y la lectura de valores de estos elementos se realiza usando los métodos llamados método-set y método-get, los cuales se definen con el modificador de acceso public.

Ejemplo:

```
class CPerson
  {
protected:
   string            m_name;                    // nombre
public:
   void              SetName(string n){m_name=n;}// establece el nombre
   string            GetName(){return (m_name);} // devuelve el nombre
  };
```

Este enfoque ofrece varias ventajas. Primero, por el nombre de la función se puede entender qué es lo

que hace: establece o recibe el valor de un elemento de clase. Segundo, tal vez en el futuro tengamos necesidad de cambiar el tipo de variable m_name en la misma clase CPerson o en alguna de sus clases derivadas.

En este caso, bastará con cambiar la implementación de la función SetName() y GetName(), mientras que los objetos de la clase CPerson se podrá usar en el programa sin ningunas modificaciones en el código, porque el usuario ni siquiera va a enterarse de que el tipo de datos m_name se haya cambiado.

Ejemplo:

```
struct Name
  {
   string            first_name;                  // nombre
   string            last_name;                   // apellido
  };

class CPerson
  {
protected:
   Name              m_name;                      // nombre
public:
   void              SetName(string n);
   string            GetName(){return(m_name.first_name+" "+m_name.last_name);}
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);
```

```
}
```

Véase también

Tipos de datos

Herencia

La particularidad de POO es la promoción de reuso del código utilizando los mecanismos de herencia. Una clase nueva se crea de una ya existente que se llama la clase base (superclase). La clase derivada (subclase) utiliza los elementos de la clase base. La clase derivada usa los elementos de la clase base, aunque también puede modificar y completarlos.

Muchos tipos son variaciones de los temas ya existentes. A menudo resulta muy agotador elaborar un código nuevo para cada uno de ellos. Además, un código nuevo supone errores nuevos. La clase derivada hereda la descripción de la clase base, siendo innecesario volver a crear y comprobar el código. Las relaciones de herencia se representan en una forma jerárquica.

La jerarquía es un método que permite copiar los elementos en toda su diversidad y complejidad. Ella introduce la clasificación de objetos. Por ejemplo, en la tabla periódica de los elementos hay gases. Ellos poseen propiedades inherentes a todos los elementos del sistema.

Los gases inertes es la siguiente subclase importante. La jerarquía consiste en que el gas inerte, como por ejemplo el argón, es un gas, y en su lugar el gas es un elemento del sistema. Este tipo de jerarquía permite explicar fácilmente el comportamiento de los gases inertes. Sabemos que sus átomos contienen protones y electrones, lo que es cierto para los demás elementos.

Sabemos que se encuentran en el estado gaseoso con la temperatura interior, igual que todos los gases. Sabemos que ningún gas de la subclase de gases inertes entra en la reacción química común con ninguno de los elemento, y esta es la propiedad de todos los gases inertes.

Vamos a ver la jerarquía en el ejemplo de las figuras geométricas. Para describir la variedad de figuras simples (círculo, triángulo, rectángulo, cuadrado, etc.) es mejor crear una clase base (TDA), que resulta ser el antecesor de todas las clases derivadas.

Vamos a crear la clase base CShape que contiene sólo los miembros más comunes que describen la figura. Estos miembros describen las propiedades características de cualquier figura, es decir, tipo de figura y las coordenadas del punto principal de enlace.

Ejemplo:

```
//--- Clase base Figura
class CShape
  {
protected:
   int       m_type;                    // tipo de figura
   int       m_xpos;                    // X - coordenada del punto de enlace
   int       m_ypos;                    // Y - coordenada del punto de enlace
public:
             CShape(){m_type=0; m_xpos=0; m_ypos=0;} // constructor
   void      SetXPos(int x){m_xpos=x;} // establezcamos X
   void      SetYPos(int y){m_ypos=y;} // establezcamos Y
  };
```

Luego vamos a crear nuevas clases derivadas de la clase base en las cuales vamos a añadir campos necesarios que especifican cada clase en particular. Para la figura Circle(círculo) hace falta añadir un miembro que contiene el valor del radio. La figura Quadrate (cuadrado) se caracteriza por el valor del lado del cuadrado. Por lo tanto, las clases derivadas, heredadas de la clase base CShape, serán

declaradas como sigue:

```
//--- clase derivada Círculo
class CCircle : public CShape        // después de dos puntos definimos la clase base
  {                                    // de la que se hace la herencia
private:
   int             m_radius;           // radio del círculo

public:
                   CCircle(){m_type=1;}// constructor, el tipo es igual a 1
  };
```

Para el cuadrado la declaración de la clase es similar:

```
//--- clase derivada Cuadrado
class CSquare : public CShape        // después de dos puntos definimos la clase base
  {                                    // de la que se hace la herencia
private:
   int             m_square_side;      // lado del cuadrado

public:
                   CSquare(){m_type=2;} // constructor, el tipo es igual a 2
  };
```

Cabe señalar que durante la creación de un objeto primero se llama al constructor de la clase base, y luego al constructor de la clase derivada. Cuando se destruye un objeto, primero se llama al destructor de la clase derivada, y luego al destructor de la clase base.

De esa manera, al declarar en la clase base los miembros más generales, podemos añadir miembros adicionales en las clases derivadas, los que precisan una clase en concreto. La herencia permite crear potentes bibliotecas de código que pueden ser utilizadas varias veces y en repetidas ocasiones.

La sintaxis de crear una clase derivada de una ya existente es como sigue:

```
class nombre_de_clase :
          (public | protected | private) opt  nombre_de_clase_base
  {
    declaración de miembros
  };
```

Uno de los aspectos de una clase derivada es la visibilidad (abertura) de sus miembros-herederos. Las palabras claves public, protected y private se utilizan para indicar el nivel de acceso de los elementos de la clase base para la clase derivada. El uso de la palabra clave private seguida de dos puntos en el encabezado de la clase derivada significa que los miembros protegidos y abiertos (protected y public) de la clase base CShape deberían ser heredados como elementos protegidos y abiertos de la clase derivada CCircle.

Los miembros privados de la clase base no están disponibles para la clase derivada. La herencia pública también significa que las clases derivadas (CCircle y CSquare) son CShape. Es decir, el cuadrado (CSquare) es la figura (CShape) pero la figura no tiene que ser obligatoriamen6te un cuadrado.

La clase derivada es una modificación de la clase base; hereda los miembros protegidos y públicos de la clase base. Los constructores y destructores de la clase base no pueden ser heredados. A menudo a la clase derivada se añaden nuevos miembros como complemento a los miembros de la clase base.

La clase derivada puede incluir la implementación de las funciones-miembro diferente de la clase base. Eso no tiene nada que ver con la sobrecarga cuando el significado del nombre de la función puede ser distinto para diferentes signaturas.

Con la herencia protegida los miembros públicos y protegidos de la clase base se hacen miembros protegidos de la clase derivada. Con la herencia privada los miembros públicos y protegidos de la clase base se hacen miembros privados de la clase derivada.

Con la herencia protegida y privada la relación de que "un objeto de clase derivada es un objeto de clase base" no es cierta. La herencia protegida y privada se encuentran muy raras veces, y hay que usar cada una de ellas con mucho cuidado.

Hay que entender que el tipo de herencia (public, protected o private) de ninguna manera influye en los modos de **acceso a los miembros de las clases base en la jerarquía de la herencia desde una clase derivada (un heredero)**. Sea como sea el tipo de herencia, desde las clases derivadas estarán disponibles sólo los miembros de la clase base declarados con los especificadores de acceso public y protected. Veremos lo arriba mencionado en un ejemplo:

```
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//+------------------------------------------------------------------+
//| Una clase de ejemplo con varios tipos de acceso                  |
//+------------------------------------------------------------------+
class CBaseClass
  {
private:             //--- el miembro privado no está disponible desde las clases der
   int               m_member;
protected:           //--- el modo protegido está disponible desde la clase base y su
   int               Member(){return(m_member);}
public:              // el constructor de la clase está disponible para todos
                     CBaseClass(){m_member=5;return;};
private:             // el modo cerrado para asignar valores al miembro m_member
   void              Member(int value) { m_member=value;};

  };
//+------------------------------------------------------------------+
//|  clase derivada con errores                                      |
//+------------------------------------------------------------------+
class CDerivaed: public CBaseClass // se puede omitir la especificación de la herenci
  {
public:
   void Func() // definimos en la clase derivada una función con las llamadas a los m
     {
      //--- intento de modificación de un miembro privado de la clase base
      m_member=0;        // error, el miembro privado de la clase base no está dispon
```

```
      Member(0);          // error, el método privado de la clase base no está disponi
      //--- lectura del miembro de la clase base
      Print(m_member);    // error, el miembro privado de la clase base no está dispon
      Print(Member());    // no hay error, el método público de la clase base está dis
     }
  };
```

En el ejemplo citado, la clase CBaseClass tiene sólo un método público, es el constructor. Los constructores se llaman automáticamente cuando se crea un objeto de la clase. Por eso no hay ninguna manera de llamar al miembro privado m_member, ni tampoco al método protegido Member() desde fuera. Pero en caso de la herencia pública (public), el método Member() de la clase base estará disponible desde las clases derivadas.

En caso de la herencia protegida (protected), todos los miembros de la clase base con el acceso público y protegido se hacen protegidos. Esto significa que si los miembros-datos y métodos públicos de la clase base estaban disponibles desde fuera, entonces ahora en caso de la herencia protegida ellos estarán disponibles sólo desde las clases del heredero y de sus posteriores clases derivadas.

```
//+------------------------------------------------------------------+
//| Una clase de ejemplo con varios tipos de acceso                  |
//+------------------------------------------------------------------+
class CBaseMathClass
  {
private:             //--- el miembro privado no está disponible desde las clases der
   double            m_Pi;
public:              //--- obtención y establecimiento del parámetro para m_Pi
   void              SetPI(double v){m_Pi=v;return;};
   double            GetPI(){return m_Pi;};
public:              // el constructor de la clase está disponible para todos
                     CBaseMathClass() {SetPI(3.14);  PrintFormat("%s",__FUNCTION__);}
  };
//+------------------------------------------------------------------+
//| Una clase derivada en la que ya no se puede modificar m_Pi       |
//+------------------------------------------------------------------+
class CProtectedChildClass: protected CBaseMathClass // herencia protegida
  {
private:
   double            m_radius;
public:              //--- métodos públicos en la clase derivada
   void              SetRadius(double r){m_radius=r; return;};
   double            GetCircleLength(){return GetPI()*m_radius;};
  };
//+------------------------------------------------------------------+
//|  Función del inicio del script                                   |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- durante la creación de una clase derivada, el constructor de la clase base se l
   CProtectedChildClass pt;
//--- especificamos el radio
```

```
   pt.SetRadius(10);
   PrintFormat("Length=%G",pt.GetCircleLength());
//--- si comentamos la cadena de abajo, obtendremos el error en la fase de compilació
// pt.SetPI(3);

//--- ahora declararemos una variable de la clase base e intentaremos establecer la c
   CBaseMathClass bc;
   bc.SetPI(10);
//--- veremos lo que ha salido
   PrintFormat("bc.GetPI()=%G",bc.GetPI());
  }
```

En este ejemplo se muestra que los métodos SetPI() y GetPi() en la clase base CBaseMathClass son públicos y están disponibles para la llamada desde cualquier parte del programa. Pero al mismo tiempo, para su derivada CProtectedChildClass las llamadas a estos métodos se puede realizar sólo desde los métodos de la misma clase CProtectedChildClass o sus clases derivadas.

En caso de la herencia privada (private), todos los miembros de la clase base con el acceso public y protected se hacen privados, y en caso de posteriores herencias resulta imposible llamarlos.

En MQL5 no hay herencia múltiple.

Véase también

Estructuras y clases

Polimorfismo

Polimorfismo es una posibilidad para objetos de diferentes clases vinculados por medio de la herencia reaccionar de varias maneras a la hora de dirigirse a la misma función-elemento. Eso permite crear unos mecanismos universales que describen el comportamiento no sólo de la clase base, sino el de las clases descendentes.

Vamos a seguir desarrollando la clase base CShape donde definimos la función miembro GetArea() que sirve para calcular la superficie de una figura. En todas las clases descendientes de la clase base vamos a redefinir esta función de acuerdo con las normas de cálculo de la superficie de una figura en concreto.

Para el cuadrado (clase CSquare) la superficie se calcula por los lados, para el círculo (clase CCircle) la superficie se calcula por el radio y etc. Podemos crear un array para almacenar los objetos del tipo CShape en el que podremos almacenar tanto el objeto de la clase base, como el de todos sus descendientes. En el futuro podremos llamar a la misma función para cualquier elemento de este array.

Ejemplo:

```
//--- Clase base
class CShape
  {
protected:
   int            m_type;                    // tipo de figura
   int            m_xpos;                    // X - coordenada del punto de enlace
   int            m_ypos;                    // Y - coordenada del punto de enlace
public:
   void           CShape(){m_type=0;};       // constructor, el tipo es igual a cero
   int            GetType(){return(m_type);};// devuelve el tipo de figura
virtual
   double         GetArea(){return (0); }    // devuelve la superficie de figura
  };
```

Ahora todas las clases derivadas tienen la función miembro getArea() que devuelve el valor cero. La implementación de esta función va a ser distinta en cada un de los descendientes.

```
//--- clase derivada Círculo
class CCircle : public CShape               // después de dos puntos definimos la cla
  {                                         // de la que se hace la herencia
private:
   double         m_radius;                 // radio del círculo

public:
   void           CCircle(){m_type=1;};     // constructor, el tipo es igual a 1
   void           SetRadius(double r){m_radius=r;};
   virtual double GetArea(){return (3.14*m_radius*m_radius);}// superficie del circul
  };
```

Para el cuadrado la declaración de la clase es similar:

```
//--- clase derivada Cuadrado
```

```
class CSquare : public CShape              // después de dos puntos definimos la clas
  {                                        // de la que se hace la herencia
private:
   double          m_square_side;          // lado del cuadrado

public:
   void            CSquare(){m_type=2;};   // constructor, el tipo es igual a 2
   void            SetSide(double s){m_square_side=s;};
   virtual double GetArea(){return (m_square_side*m_square_side);}//superficie del cu
  };
```

Como para el cálculo de la superficie del círculo y cuadrado se necesitan los valores correspondientes de los miembros m_radius y m_square_side, entonces en la declaración de la clase correspondiente hemos añadido las funciones SetRadius y SetSide().

Se supone que en nuestro programa se utilizan los objetos de deferentes tipos (CCircle y CSquare) pero heredados de un tipo base CShape. El polimorfismo no permite crear un array de objetos del tipo base CShape pero durante la declaración de este array los objetos aún no se conoces y su tipo no está definido.

La decisión sobre el objeto de qué tipo va a contenerse en cada elemento del array va a tomarse directamente en el proceso de ejecución del programa. Esto supone la creación dinámica de objetos de las clases correspondientes, y por consecuencia, la necesidad de usar los punteros a objetos en vez de los objetos en sí.

Para la creación dinámica de los objetos se utiliza el operador new. Cada objeto tiene que eliminarse de manera individual y explícita con el operador delete. Por eso declararemos un array de punteros del tipo CShape, y crearemos para cada elemento suyo un objeto del tipo necesario (**new Nombre _de_la_clase)**, tal como se muestra en el ejemplo del script:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- declararemos un array de punteros del tipo base
   CShape *shapes[5];   // array de punteros a objetos CShape

//--- aquí llenamos el array con objetos derivados
//--- declararemos un puntero al objeto del tipo CCircle
   CCircle *circle=new CCircle();
//--- establecemos las propiedades del objeto según el puntero circle
   circle.SetRadius(2.5);
//--- colocaremos el valor del puntero en shapes[0]
   shapes[0]=circle;

//--- crearemos otro objeto CCircle más y escribiremos su puntero en shapes[1]
   circle=new CCircle();
   shapes[1]=circle;
   circle.SetRadius(5);
```

```
//--- aquí "hemos olvidado" establecer aposta el valor para shapes[2]
//circle=new CCircle();
//circle.SetRadius(10);
//shapes[2]=circle;

//--- estableceremos el valor NULL para el elemento que no se utiliza
   shapes[2]=NULL;

//--- crearemos el objeto CSquare y escribiremos su puntero en shapes[3]
   CSquare *square=new CSquare();
   square.SetSide(5);
   shapes[3]=square;

//--- crearemos el objeto CSquare y escribiremos su puntero en shapes[4]
   square=new CSquare();
   square.SetSide(10);
   shapes[4]=square;

//--- tenemos un array de punteros, obtendremos su tamaño
   int total=ArraySize(shapes);
//--- pasaremos en un ciclo por todos los punteros en el array
   for(int i=0; i<5;i++)
     {
      //--- si el puntero en el índice especificado es válido
      if(CheckPointer(shapes[i])!=POINTER_INVALID)
        {
         //--- mostraremos en el log el tipo y la superficie del figura
         PrintFormat("El objeto del tipo %d tiene la superficie %G",
               shapes[i].GetType(),
               shapes[i].GetArea());
        }
      //--- si el puntero tiene el tipo POINTER_INVALID
      else
        {
         //--- avisaremos sobre el error
         PrintFormat("¡El objeto shapes[%d] no ha sido inicializado! Su puntero es %s
                     i,EnumToString(CheckPointer(shapes[i])));
        }
     }

//--- tenemos que eliminar personalmente todos los objetos dinámicos creados
   for(int i=0;i<total;i++)
     {
      //--- se puede eliminar sólo los objetos cuyo puntero tiene el tipo POINTER_DYN
      if(CheckPointer(shapes[i])==POINTER_DYNAMIC)
        {
         //--- avisaremos sobre la eliminación
         PrintFormat("Eliminamos shapes[%d]",i);
```

```
         //--- eliminaremos los objetos según su puntero
         delete shapes[i];
        }
     }
  }
```

Preste la atención a que cuando se elimina un objeto por el operador delete, hay que comprobar el tipo de su puntero. A través de delete se puede eliminar sólo los objetos que tienen los punteros POINTER_DYNAMIC. Para los punteros de otros tipos se mostrará el error.

Pero a parte de la redefinición de la función durante la herencia, el polimorfismo incluye también la implementación de la misma función con diferentes conjuntos de parámetros dentro de los límites de una clase. Eso significa que la clase puede tener varias funciones con el mismo nombre pero de diferentes tipos y/o conjunto de parámetros. En este caso el polimorfismo se implementa a través de la sobrecarga de funciones.

Véase también

Biblioteca estándar

Sobrecarga

Dentro de los límites de una clase se puede definir dos o más métodos que conjuntamente usan el mismo nombre pero tienen diferente cantidad de parámetros. Cuando eso ocurre, los métodos se llaman *sobrecargados*, y del proceso se dice como de la *sobrecarga de método*. La sobrecarga de método es uno de los modos de realizar el polimorfismo. La sobrecarga de método en las clases se efectua según las mismas reglas que la sobrecarga de función.

Si no hay correspondencia exacta para la función llamada, el compilador busca la función conveniente en tres niveles de una manera consecutiva:

1. búsqueda entre los métodos de la clase;
2. búsqueda entre los métodos de las clases base, del ascendiente más próximo hasta el primero sucesivamente;
3. búsqueda entre las demás funciones.

Si no se ha encontrado la correspondencia exacta en ninguno de los niveles, pero hay unas funciones convenientes en los niveles diferentes, entonces se utiliza la función en el nivel menor. No puede haber más de una función conveniente dentro de los límites de un nivel.

En MQL5 no hay sobrecarga de operadores.

Véase también

Sobrecarga de funciones

Funciones virtuales

La palabra clave virtual es un especificador de la función que proporciona el mecanismo para la elección dinámica durante la fase de ejecución de una función miembro conveniente entre las funciones de la clase base y derivada. Las estructuras no pueden tener funciones virtuales. Puede usarse para el cambio de las declaraciones sólo de las funciones miembro.

La función virtual, igual que una corriente, debe tener un cuerpo ejecutable. Durante la llamada su semántica es la misma que la de las demás funciones.

Una función virtual puede ser sustituida en una clase derivada. La elección de qué tipo de definición de la función llamar para una función virtual, se hace de una forma dinámica (durante la fase de ejecución). Un caso típico es cuando la clase base contiene una función virtual, y las clases derivadas tienen sus versiones de esta función.

El puntero a la clase base puede indicar al objeto de la clase base o al objeto de la clase derivada. La elección de la función miembro va a ser realizada en la fase de ejecución y va a depender del tipo del objeto y no del tipo del puntero. Si no hay miembro del tipo derivado, por defecto se utiliza la función virtual de la clase base.

Los destructores siempre son virtuales, independientemente de que si están declarados con la palabra clave virtual o no.

Vamos a ver el uso de funciones virtuales en el ejemplo del programa MT5_Tetris.mq5. La clase base CTetrisShape con la función virtual Draw (dibujar) está definida en el archivo incluido MT5_TetisShape.mqh.

```
//+------------------------------------------------------------------+
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(++m_turn>3) r
   virtual void      Draw()                      { return;                 }
   virtual bool      CheckDown(int& pad_array[]);
   virtual bool      CheckLeft(int& side_row[]);
   virtual bool      CheckRight(int& side_row[]);
  };
```

Luego, para cada clase derivada, esta función se implementa de acuerdo con las particularidades de la clase descendiente. Por ejemplo, la primera figura CTetrisShape1 tiene su propia implementación de la función Draw():

```
class CTetrisShape1 : public CTetrisShape
  {
public:
   //--- dibujo de figura
   virtual void      Draw()
     {
      int    i;
      string name;
      //---
      if(m_turn==0 || m_turn==2)
        {
         //--- vara 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
        {
         //--- vara 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);
           }
        }
     }
  }
```

La figura cuadrado está descrita por la clase CTetrisShape6 y tiene su propia implementación del método Draw():

```
class CTetrisShape6 : public CTetrisShape
  {
public:
   //--- dibujo de figura
```

```
   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);
        }
     }
  };
```

Dependiendo de qué clase ha sido creado el objeto, se llama a la función virtual de una u otra clase derivada.

```
void CTetrisField::NewShape()
  {
//--- creamos una de 7 posibles figuras de una manera aleatoria
   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;
     }
//--- dibujamos
   m_shape.Draw();
//---
  }
```

Véase también

Biblioteca estándar

Miembros estáticos de una clase/estructura

Miembros estáticos

Los miembros de una clase pueden ser declarados con el uso del modificador de la clase de memoria static. Estos miembros de datos se comparten por todos los ejemplares de esta clase y se guardan en un sitio. Los miembros de datos no estáticos se crean para cada variable-objeto de la clase.

La imposibilidad de declarar los miembros estáticos de una clase conduciría a la necesidad de declarar estos datos a nivel global del programa. Esto rompería las relaciones entre los datos y su clase, además de no concordarse con el paradigma básica de la POO - la unión de datos y métodos para su procesamiento dentro de una clase. El miembro estático permite existir a los datos de la clase que no son específicos para un ejemplar particular, en el campo de alcance de la clase.

Puesto que un miembro estático de la clase no depende de un ejemplar concreto, la referencia a él es como sigue:

```
class_name::variable
```

donde *class_name* es el nombre de la clase, y *variable* significa el nombre del miembro de la clase.

Como ve, para acceder al miembro estático de la clase, se utiliza el operador de resolución de contexto ::. Cuando accedemos a un miembro estático dentro de los métodos de la clase, el operador de contexto es opcional.

No es necesario inicializar los miembros estáticos de una clase explícitamente a nivel global, ya que serán inicializados automáticamente cuando se inicia el programa. El miembro estático del tipo entero y real se inicializa automáticamente con un cero, al tipo string se le asigna el valor NULL. Para los miembros estáticos de tipos complejos se llama el constructor por defecto, y si no lo hay, el constructor con los parámetros predefinidos.

El miembro estático de la clase puede ser inicializado explícitamente con un valor necesario. Para eso tiene que ser definido e inicializado a nivel global. Por ejemplo, tenemos una clase *CParser* que se utiliza para el análisis sintáctico de textos, y necesitamos obtener el número total de palabras y símbolos procesados. Sólo hay que declarar los miembros necesarios de la clase como estáticos e inicializarlos a nivel global. Entonces, todos los ejemplares de la clase van a utilizar durante su trabajo los contadores comunes de palabras y símbolos.

```
//+------------------------------------------------------------------+
//|  Clase "Analizador de textos"                                    |
//+------------------------------------------------------------------+
class CParser
  {
public:
   static int        s_words;
   static int        s_symbols;
   //--- constructor y destructor
                     Parser(void);
                    ~Parser(void){};
  };
...
//--- inicialización de los miembros estáticos de la clase Parser a nivel global
int CParser::s_words=0;
int CParser::s_symbols=0;
```

Un miembro estático de la clase puede ser declarado con la palabra clave *const*. Estas constantes estáticas tienen que ser inicializadas a nivel global con la palabra clave *const:*

```
//+------------------------------------------------------------------+
//|  Clase "Pila" para almacenar los datos procesados                |
//+------------------------------------------------------------------+
class CStack
  {
public:
                     CStack(void);
                    ~CStack(void){};
...
private:
   static const int  s_max_length; // capacidad máxima de la pila
  };

//--- inicialización de la constante estática de la clase CStack
const int CStack::s_max_length=1000;
```

Puntero this

La palabra clave this denota un puntero declarado implícitamente a sí mismo - a un ejemplar concreto de la clase en contexto del cual se ejecuta el método. Se puede utilizarlo sólo en los métodos no estáticos de la clase. El puntero this es un miembro implícito no estático de cualquier clase.

En las funciones estáticas se puede acceder sólo a los miembros/métodos estáticos de la clase.

Métodos estáticos

En MQL5 se permite utilizar las funciones-miembros del tipo static. El modificador *static* debe ir antes del tipo de la función devuelto en la declaración dentro de la clase.

```
class CStack
  {
public:
   //--- constructor y destructor
                     CStack(void):m_capacity(1000){};
                    ~CStack(void){};
   //--- capacidad máxima de la pila
   static int        Capacity();
private:
   int               m_length;     // número de elementos en la pila
   static const int  s_max_length; // capacidad máxima de la pila
  };
//+------------------------------------------------------------------+
//| Devuelve el número máximo de elementos para el almacenamiento en la pila|
//+------------------------------------------------------------------+
int CStack::Capacity(void)
  {
   return(s_max_length);
  }
```

El método con modificador const se llama constante y no puede modificar los miembros implícitos de su clase. La declaración de las funciones constantes de la clase y los parámetros constantes se llama *control de constancia* (const-correctness). Gracias a este control se puede estar seguro de que el compilador va a controlar la constancia de valores de los objetos y mostrará error en la fase de compilación en caso de violación.

El modificador const se pone después de la lista de argumentos dentro de la declaración de la clase. La definición fuera de la clase también debe incluir el modificador *const*:

```
//+------------------------------------------------------------------+
//| Clase "Rectángulo"                                               |
//+------------------------------------------------------------------+
class CRectangle
  {
private:
   double            m_width;      // ancho
   double            m_height;     // alto
public:
   //--- constructores y destructor
                     CRectangle(void):m_width(0),m_height(0){};
                     CRectangle(const double w,const double h):m_width(w),m_height(h)
                    ~CRectangle(void){};
   //--- cálculo de la superficie
   double            Square(void) const;
   static double     Square(const double w,const double h);// { return(w*h); }
  };
//+------------------------------------------------------------------+
//|  Devuelve la superficie del objeto "Rectángulo"                  |
//+------------------------------------------------------------------+
double CRectangle::Square(void) const
  {
   return(Square(m_width,m_height));
  }
//+------------------------------------------------------------------+
//|  Devuelve el resultado de multiplicación de dos variables
//+------------------------------------------------------------------+
static double CRectangle::Square(const double w,const double h)
  {
   return(w*h);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- creamos un rectángulo rect con los lados 5 y 6
   CRectangle rect(5,6);
//--- buscaremos la superficie del rectángulo utilizando el método constante
   PrintFormat("rect.Square()=%.2f",rect.Square());
//--- buscaremos el resultado de los números utilizando el método estático de la clas
   PrintFormat("CRectangle::Square(2.0,1.5)=%f",CRectangle::Square(2.0,1.5));
  }
```

Como argumento adicional a favor del uso del control de constancia sirve el hecho de que el compilador en este caso genera una optimización especial. Por ejemplo, coloca el objeto constante en la memoria sólo para la lectura.

Una función estática no puede ser determinada con el modificador const, puesto que este modificador garantiza la constancia de los miembros del ejemplar cuando se llama a esta función. Pero, como ya

hemos dicho antes, una función estática por definición no puede acceder a los miembros no estáticos de la clase.

Véase también

Variables estáticas, Variables, Enlaces. Modificador & y la palabra clave this

Plantillas de funciones

Las funciones sobrecargadas suelen utilizarse para la ejecución de operaciones semejantes con diferentes tipos de datos. La función ArraySize() es un sencillo ejemplo en el lenguaje MQL5. Ella devuelve el tamaño del array de cualquier tipo. En realidad, esta función del sistema es una función sobrecargada, y toda la implementación de esta sobrecarga está ocultada de los desarrolladores de programas MQL5:

```
int  ArraySize(
   void&  array[]      // array a comprobar
   );
```

Es decir, en realidad para cada invocación de esta función el compilador del lenguaje MQL5 inserta una implementación necesaria. Por ejemplo, así es como se puede hacer para los arrays del tipo entero:

```
int  ArraySize(
   int&  array[]      // array con los elementos del tipo int
   );
```

Y para el array del tipo MqlRates que se usa para el trabajo con las cotizaciones en el formato de datos históricos, la función ArraySize() puede representarse de la siguiente manera:

```
int  ArraySize(
   MqlRates&  array[] // array llenado con los valores del tipo MqlRates
   );
```

De esta manera, resulta muy cómodo utilizar la misma función para trabajar con diferentes tipos. Pero hace falta realizar todo el trabajo preliminar de manera individual, es decir: sobrecargar la función necesaria para todos los tipos de datos con los que tendrá que trabajar correctamente.

Hay una solución más conveniente. Si las operaciones idénticas deben realizarse para cada uno de los tipos de datos, la solución más compacta y cómoda sería el uso de las plantillas de funciones. En este caso, al programador le hace falta escribir solamente una descripción de la plantilla de la función. Cuando la plantilla se describe de esta manera, será suficiente especificar algún parámetro formal en vez de un determinado tipo de datos con los que debe trabajar la función. El compilador va a generar automáticamente diferentes funciones para el procesamiento correspondiente de cada tipo basándose en los tipos de los argumentos utilizados durante la invocación de la función.

La definición de la plantilla de una función se empieza con la palabra clave template seguida de la lista de los parámetros formales encerrados entre los corchetes angulares. Antes de cada parámetro formal se pone la palabra clave typename. Los tipos formales de los parámetros son los tipos built-in o los tipos definidos por el usuario. Se utilizan:

- para especificar los tipos de argumentos de la función,
- para especificar los tipos del valor devuelto de la función,
- para declarar variables dentro del cuerpo de descripción de la función.

El número de parámetros de una plantilla no podrá exceder de ocho. Cada parámetro formal de la definición de la plantilla tiene que aparecer en la lista de los parámetros de la función por lo menos una vez. Cada uno de los nombres del parámetro formal tiene que ser único.

Aquí tenemos un ejemplo de una función para la búsqueda del valor máximo en el array de cualquier tipo numérico (números enteros y reales):

```
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);
  }
```

Esta plantilla define la función que busca el valor máximo en el array pasado y devuelve este valor como resultado. Vamos a recordar que la función ArrayMaximum() incorporada en MQL5 devuelve sólo el índice del valor máximo encontrado que será utilizado a continuación para obtener el mismo valor en cuestión. Por ejemplo:

```
//--- creamos un array
   double array[];
   int size=50;
   ArrayResize(array,size);
//---  lo llenamos con valores aleatorios
   for(int i=0;i<size;i++)
     {
      array[i]=MathRand();
     }

//--- buscamos la posición del elemento máximo en el array
   int max_position=ArrayMaximum(array);
//--- ahora obtenemos el mismo valor máximo en el array
   double max=array[max_position];
//--- visualizamos el valor encontrado
   Print("Max value = ",max);
```

De esta manera, hemos necesitado dos pasos para obtener el valor máximo del array. A través de la plantilla de la función ArrayMax() podemos obtener el resultado del tipo necesario con sólo pasar el array del tipo correspondiente a esta función. Es decir, en vez de las últimas dos líneas

```
//--- buscamos la posición del elemento máximo en el array
   int max_position=ArrayMaximum(array);
//--- ahora obtenemos el mismo valor máximo en el array
   double max=array[max_position];
```

ahora podemos usar una sola línea que va a devolver de una sola vez el resultado del mismo tipo que tiene el array pasado:

```
//--- buscamos el valor máximo
   double max=ArrayMax(array);
```

En este caso el tipo del resultado devuelto por la función ArrayMax() va a corresponder automáticamente al tipo del array.

Es necesario usar la palabra clave typename para obtener el tipo del argumento en forma de una cadena, con el fin de crear modos universales de trabajo con diferentes tipos de datos. Vamos a demostrarlo con el ejemplo de una función que devuelve el tipo de datos en forma de una cadena:

```
#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));
//---
  }
//+------------------------------------------------------------------+
//| El tipo se devuelve como una línea                               |
//+------------------------------------------------------------------+
template<typename T>
string GetTypeName(const T &t)
  {
//--- devolvemos el tipo en forma de una línea
   return(typename(T));
//---
  }
```

Las plantillas de las funciones también se puede utilizar para los métodos de clase, por ejemplo:

```
class CFile
  {
   ...
public:
   ...
   template<typename T>
   uint WriteStruct(T &data);
  };

template<typename T>
uint CFile::WriteStruct(T &data)
  {
   ...
   return(FileWriteStruct(m_handle,data));
  }
```

Las plantillas de las funciones no se puede declarar con las palabras clave export, virtual y #import.

Constantes estándares, enumeraciones y estructuras

Para facilitar la escritura del programa y para hacer la percepción de los textos fuentes de programas más cómoda, en el lenguaje MQL5 están previstas las constantes estándares predefinidas y las enumeraciones. Además, para almacenar la información se utilizan las estructuras auxiliares.

Las constantes estándares son similares a las macro sustituciones y tienen el tipo int.

Las constantes están agrupadas según su destinación:

- Constantes de gráficos se utilizan durante el trabajo con los gráficos de precios: apertura, navegación, fijación de parámetros;
- Constantes de objetos están destinadas para procesar los objetos gráficos que se puede crear y mostrar en los gráficos;
- Constantes de indicadores sirven para trabajar con indicadores estándares y de usuario ;
- Estado de ambiente describe las propiedades de un programa mql5, facilita la información sobre el terminal de cliente, instrumento financiero y cuenta corriente de trading;
- Constantes comerciales permiten precisar diversa información durante el proceso de comercio;
- Constantes nombradas son constantes del lenguaje MQL5;
- Estructuras de datos describen las formas utilizadas de almacenamiento de datos;
- Códigos de errores y advertencias describen mensajes del compilador y respuestas del servidor comercial a las peticiones comerciales;
- Constantes de entrada/salida están destinadas para trabajar con las funciones de archivo y para mostrar mensajes en la pantalla a través de la función MessageBox().

Constantes de gráficos

Las constantes que describen diferentes propiedades de los gráficos están divididas en siguientes grupos:

- Tipos de eventos – eventos que ocurren a la hora de trabajar con los gráficos;
- Períodos de gráficos – períodos built-in estándares;
- Propiedades de gráficos – identificadores que se utilizan como parámetros de las funciones para trabajar con gráficos;
- Constantes de posicionamiento - valor del parámetro de la función ChartNavigate();
- Visualización de gráficos - configuración de la apariencia de gráfico.

Tipos de eventos de gráfico

Existen 9 tipos de eventos que pueden ser procesados utilizando la función predefinida OnChartEvent(). Para los eventos de usuario están previstos 65536 identificadores en el rango de CHARTEVENT_CUSTOM a CHARTEVENT_CUSTOM_LAST inclusive. Para generar un evento de usuario hace falta usar la función EventChartCustom().

ENUM_CHART_EVENT

Identificador	Descripción
CHARTEVENT_KEYDOWN	Teclazos
CHARTEVENT_MOUSE_MOVE	Desplazamiento del ratón y pulsación de los botones del ratón (si para el gráfico está establecida la propiedad CHART_EVENT_MOUSE_MOVE=true)
CHARTEVENT_OBJECT_CREATE	Creación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_CREATE=true)
CHARTEVENT_OBJECT_CHANGE	Evento de cambio de propiedades de un objeto gráfico a través del diálogo de propiedades
CHARTEVENT_OBJECT_DELETE	Eliminación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_DELETE=true)
CHARTEVENT_CLICK	Clic en un gráfico
CHARTEVENT_OBJECT_CLICK	Clic en un objeto gráfico
CHARTEVENT_OBJECT_DRAG	Arrastrar un objeto gráfico
CHARTEVENT_OBJECT_ENDEDIT	Fin de edición del texto en el objeto gráfico Edit
CHARTEVENT_CHART_CHANGE	Modificación de dimensiones del gráfico o de sus propiedades a través del diálogo de propiedades
CHARTEVENT_CUSTOM	Número inicial de un evento del rango de eventos de usuario
CHARTEVENT_CUSTOM_LAST	Número final de un evento del rango de eventos de usuario

Para cada tipo del evento, los parámetros de entrada de la función OnChartEvent() tienen determinados valores que son necesarios para procesar este evento. En la tabla de abajo están especificados los eventos y valores que se pasan a través de los parámetros.

Evento	Valor del parámetro id	Valor del parámetro	Valor del parámetro	Valor del parámetro spara

		lparam	dparam	m
Evento del teclado	CHARTEVENT_KEYDOWN	código de la tecla pulsada	Número de pulsaciones de la tecla generadas mientras ésta se mantenía en estado pulsado	Valor literal de la máscara de bits que describe el estatus de las teclas del teclado
Eventos del ratón (si para el gráfico está establecida la propiedad CHART_EVENT_MOUSE_MOVE=true)	CHARTEVENT_MOUSE_MOVE	coordinada X	coordinada Y	Valor literal de la máscara de bits que describe el estatus de los botones del ratón
Evento de creación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_CREATE=true)	CHARTEVENT_OBJECT_CREATE	—	—	Nombre del objeto gráfico creado
Evento del cambio de propiedades de un objeto a través del diálogo de propiedades	CHARTEVENT_OBJECT_CHANGE	—	—	Nombre del objeto gráfico modificado
Evento de eliminación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_DELETE=true)	CHARTEVENT_OBJECT_DELETE	—	—	Nombre del objeto gráfico eliminado
Evento de cliquear sobre un gráfico	CHARTEVENT_CLICK	coordinada X	coordinada Y	—
Evento de	CHARTEVENT_OB	coordinada X	coordinada Y	Nombre del

cliquear sobre un objeto gráfico	JECT_CLICK			objeto gráfico en el que ha ocurrido un evento
Evento de mover un objeto gráfico usando el ratón	CHARTEVENT_OBJECT_DRAG	—	—	Nombre del objeto gráfico movido
Evento del fin de edición del texto en el campo de introducción del objeto gráfico "Campo de texto"	CHARTEVENT_OBJECT_ENDEDIT	—	—	Nombre del objeto gráfico "Campo de texto" donde ha finalizado la introducción del texto
Evento de modificación de dimensiones del gráfico o de sus propiedades a través del diálogo de propiedades	CHARTEVENT_CHART_CHANGE	—	—	—
Identificador del evento de usuario	CHARTEVENT_CUSTOM+N	Valor determinado por la función EventChartCustom()	Valor determinado por la función EventChartCustom()	Valor determinado por la función EventChartCustom()

Ejemplo:

```
#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("Ha sido arrancado el Asesor Experto con el nombre ",MQL5InfoString(MQL5_PRO
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,         // identificador del evento
                  const long& lparam,   // parámetro de evento del tipo long
                  const double& dparam, // parámetro de evento del tipo double
                  const string& sparam  // parámetro de evento del tipo string
                  )
  {
//--- el botón izquierdo del ratón ha sido pulsado en el gráfico
   if(id==CHARTEVENT_CLICK)
     {
      Print("Coordinadas del clic de ratón en el gráfico: x= ",lparam,"  y= ",dparam)
     }
//--- el ratón ha hecho un clic en el objeto gráfico
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      Print("Clic del ratón en el objeto con el nombre '"+sparam+"'");
     }
//--- la tecla del teclado ha sido apretada
   if(id==CHARTEVENT_KEYDOWN)
     {
      switch(lparam)
        {
         case KEY_NUMLOCK_LEFT:  Print("Apretada KEY_NUMLOCK_LEFT");   break;
         case KEY_LEFT:          Print("Apretada KEY_LEFT");           break;
         case KEY_NUMLOCK_UP:    Print("Apretada KEY_NUMLOCK_UP");     break;
         case KEY_UP:            Print("Apretada KEY_UP");             break;
         case KEY_NUMLOCK_RIGHT: Print("Apretada KEY_NUMLOCK_RIGHT");  break;
         case KEY_RIGHT:         Print("Apretada KEY_RIGHT");          break;
         case KEY_NUMLOCK_DOWN:  Print("Apretada KEY_NUMLOCK_DOWN");   break;
         case KEY_DOWN:          Print("Apretada KEY_DOWN");           break;
         case KEY_NUMPAD_5:      Print("Apretada KEY_NUMPAD_5");       break;
         case KEY_NUMLOCK_5:     Print("Apretada KEY_NUMLOCK_5");      break;
         default:                Print("Apretada una de las teclas no especificadas")
        }
      ChartRedraw();
     }
//--- el objeto ha sido eliminado
   if(id==CHARTEVENT_OBJECT_DELETE)
     {
```

```
      Print("Se ha eliminado el objeto con el nombre ",sparam);
     }
//--- el objeto ha sido creado
   if(id==CHARTEVENT_OBJECT_CREATE)
     {
      Print("Se ha creado el objeto con el nombre ",sparam);
     }
//--- el objeto ha sido arrastrado o las coordinadas de puntos de anclaje han sido ca
   if(id==CHARTEVENT_OBJECT_DRAG)
     {
      Print("Cambio de los puntos de enlace del objeto con el nombre ",sparam);
     }
//--- el texto en la casilla Edit del objeto gráfico ha sido cambiado
   if(id==CHARTEVENT_OBJECT_ENDEDIT)
     {
      Print("Ha sido cambiado el texto en el objeto Edit ",sparam);
     }
  }
```

Véase también

Funciones de procesamiento de eventos, Trabajo con eventos

Períodos de gráficos

Todos los períodos predefinidos de gráficos tienen sus identificadores únicos. El identificador PERIOD_CURRENT significa el período corriente del gráfico en el cual se inicia el programa mql5.

ENUM_TIMEFRAMES

Identificador	Descripción
PERIOD_CURRENT	Período corriente
PERIOD_M1	1 minuto
PERIOD_M2	2 minutos
PERIOD_M3	3 minutos
PERIOD_M4	4 minutos
PERIOD_M5	5 minutos
PERIOD_M6	6 minutos
PERIOD_M10	10 minutos
PERIOD_M12	12 minutos
PERIOD_M15	15 minutos
PERIOD_M20	20 minutos
PERIOD_M30	30 minutos
PERIOD_H1	1 hora
PERIOD_H2	2 horas
PERIOD_H3	3 horas
PERIOD_H4	4 horas
PERIOD_H6	6 horas
PERIOD_H8	8 horas
PERIOD_H12	12 horas
PERIOD_D1	1 día
PERIOD_W1	1 semana
PERIOD_MN1	1 mes

Ejemplo:

```
   string chart_name="test_Object_Chart";
   Print("Vamos a intentar crear un objeto Chart con el nombre ",chart_name);
//--- si este objeto no existe, lo creamos
   if(ObjectFind(0,chart_name)<0)ObjectCreate(0,chart_name,OBJ_CHART,0,0,0,0,0);
```

```
//--- definimos el símbolo
   ObjectSetString(0,chart_name,OBJPROP_SYMBOL,"EURUSD");
//--- determinamos la coordinada X para el punto de anclaje
   ObjectSetInteger(0,chart_name,OBJPROP_XDISTANCE,100);
//--- determinamos la coordinada Y para el punto de anclaje
   ObjectSetInteger(0,chart_name,OBJPROP_YDISTANCE,100);
//--- establecemos el ancho
   ObjectSetInteger(0,chart_name,OBJPROP_XSIZE,400);
//--- establecemos el alto
   ObjectSetInteger(0,chart_name,OBJPROP_YSIZE,300);
//--- definimos el período
   ObjectSetInteger(0,chart_name,OBJPROP_PERIOD,PERIOD_D1);
//--- determinamos la escala (de 0 a 5)
   ObjectSetDouble(0,chart_name,OBJPROP_SCALE,4);
//--- deshabilitamos la selección con el ratón
   ObjectSetInteger(0,chart_name,OBJPROP_SELECTABLE,false);
```

Véase también

PeriodSeconds, Period, Fecha y hora, Visibilidad de objetos

Propiedades de gráficos

Los identificadores de la familia de enumeraciones ENUM_CHART_PROPERTY se usan como parámetros de las funciones para trabajar con gráfico. Las siglas r/o en la columna "Tipo de la propiedad" significa que dicha propiedad está destinada únicamente para la lectura y no puede ser cambiada. Las siglas w/o en la columna "Tipo de la propiedad" significa que dicha propiedad está destinada únicamente para la escritura y no puede ser obtenida. A la hora de acceder a algunas propiedades, es necesario indicar el parámetro-modificador adicional (modifier) que sirve para especificar el número de subventana del gráfico. 0 significa la ventana principal.

Para las funciones ChartSetInteger() y ChartGetInteger()

ENUM_CHART_PROPERTY_INTEGER

Identificador	Descripción	Tipo de la propiedad
CHART_IS_OBJECT	Indicio para la identificación del objeto "Gráfico" (OBJ_CHART) - para un objeto gráfico devuelve true. Para el gráfico real devuelve false	bool r/o
CHART_BRING_TO_TOP	Visualización del gráfico por encima de los demás	bool w/o
CHART_MOUSE_SCROLL	Desplazamiento del gráfico por la horizontal usando el botón izquierdo del ratón. Desplazamiento por la vertical también estará disponible si el valor de cualquiera de las siguientes tres propiedades está establecido en true: CHART_SCALEFIX, CHART_SCALEFIX_11 o CHART_SCALE_PT_PER_BAR	bool
CHART_EVENT_MOUSE_MOVE	Envía una notificación sobre los eventos de desplazamiento y pulsación de botones del ratón (CHARTEVENT_MOUSE_MOVE) a todos los programas mql5 presentes en el gráfico	bool
CHART_EVENT_OBJECT_CREATE	Envía una notificación sobre el evento de creación de nuevo objeto gráfico (CHARTEVENT_OBJECT_CREATE) a todos los programas mql5 presentes en el gráfico	bool

CHART_EVENT_OBJECT_DELETE	Envía una notificación sobre el evento de eliminación de un objeto gráfico (CHARTEVENT_OBJECT_DELETE) a todos los programas mql5 presentes en el gráfico	bool
CHART_MODE	Tipo de gráfico (velas, barras o líneas)	enum ENUM_CHART_MODE
CHART_FOREGROUND	Gráfico de precio en el fondo	bool
CHART_SHIFT	Modo de sangría del gráfico de precio desde el lado derecho	bool
CHART_AUTOSCROLL	Modo automático de traspaso al lado derecho del gráfico	bool
CHART_SCALE	Escala	int de 0 a 5
CHART_SCALEFIX	Modo de escala fija	bool
CHART_SCALEFIX_11	Modo de escala 1:1	bool
CHART_SCALE_PT_PER_BAR	Modo de especificar la escala en puntos por barra	bool
CHART_SHOW_OHLC	Muestra de valores OHLC en esquina izquierda superior	bool
CHART_SHOW_BID_LINE	Muestra del valor Bid con una línea horizontal en el gráfico	bool
CHART_SHOW_ASK_LINE	Muestra del valor Ask con una línea horizontal en el gráfico	bool
CHART_SHOW_LAST_LINE	Muestra del valor Last con una línea horizontal en el gráfico	bool
CHART_SHOW_PERIOD_SEP	Muestra de separadores verticales entre períodos adyacentes	bool
CHART_SHOW_GRID	Muestra de rejilla en el gráfico	bool
CHART_SHOW_VOLUMES	Muestra de volúmenes en el gráfico	enum ENUM_CHART_VOLUME_MODE
CHART_SHOW_OBJECT_DESCR	Descripciones emergentes de objetos gráficos	bool
CHART_VISIBLE_BARS	Cantidad de barras en el gráfico disponibles para ser mostrados	int r/o
CHART_WINDOWS_TOTAL	Número total de las ventanas del gráfico, incluyendo las	int r/o

	subventanas de indicadores	
CHART_WINDOW_IS_VISIBLE	Visibilidad de subventanas	bool r/o modificador - número de subventana
CHART_WINDOW_HANDLE	Manejador (handle) del gráfico (HWND)	int r/o
CHART_WINDOW_YDISTANCE	Distancia en píxeles por el eje vertical Y entre el marco superior de la subventana del indicador y el marco superior de la ventana principal del gráfico. En caso del evento del ratón, las coordinadas del cursor se pasan en términos de las coordinadas de la ventana principal del gráfico, mientras que las coordinadas de los objetos gráficos en la subventana del indicador se establecen respecto a la esquina superior izquierda de la subventana. El valor es necesario para convertir las coordinadas absolutas del gráfico principal a las coordinadas locales de la subventana para un trabajo correcto con los objetos gráficos cuyas coordinadas se establecen respecto a la esquina superior izquierda del cuadro de la subventana.	int r/o modificador - número de subventana
CHART_FIRST_VISIBLE_BAR	Número de la primera barra visible en el gráfico. Indexación de las barras corresponde a la serie temporal.	int r/o
CHART_WIDTH_IN_BARS	El ancho del gráfico en barras	int r/o
CHART_WIDTH_IN_PIXELS	El ancho del gráfico en píxeles	int r/o
CHART_HEIGHT_IN_PIXELS	El alto del gráfico en píxeles	int modificador - número de subventana
CHART_COLOR_BACKGROUND	Color del fondo del gráfico	color
CHART_COLOR_FOREGROUND	Color de ejes, escala y línea OHLC	color
CHART_COLOR_GRID	Color de rejilla	color

CHART_COLOR_VOLUME	Color de volúmenes y niveles de apertura de posiciones	color
CHART_COLOR_CHART_UP	Color de barra al alza, sombras y bordes del cuerpo de la vela de toro	color
CHART_COLOR_CHART_DOWN	Color de barra a la baja, sombras y bordes del cuerpo de la vela de oso	color
CHART_COLOR_CHART_LINE	Color de la línea del gráfico y de las velas japonesas "Doji"	color
CHART_COLOR_CANDLE_BULL	Color del cuerpo de la vela toro	color
CHART_COLOR_CANDLE_BEAR	Color del cuerpo de la vela oso	color
CHART_COLOR_BID	Color de la línea Bid-precio	color
CHART_COLOR_ASK	Color de la línea Ask-precio	color
CHART_COLOR_LAST	Color de la línea de precio de la última transacción realizada (Last)	color
CHART_COLOR_STOP_LEVEL	Color de los niveles de stop-órdenes (Stop Loss y Take Profit)	color
CHART_SHOW_TRADE_LEVELS	Visualiza en el gráfico los niveles comerciales (niveles de posiciones abiertas, Stop Loss, Take Profit y ordenes pendientes)	bool
CHART_DRAG_TRADE_LEVELS	Permiso para arrastrar los niveles de trading por el gráfico utilizando el ratón. Por defecto, el modo de arrastre está activado (valor true)	bool
CHART_SHOW_DATE_SCALE	Visualiza la escala de tiempo en el gráfico	bool
CHART_SHOW_PRICE_SCALE	Visualiza la escala de precios en el gráfico	bool

Para las funciones ChartSetDouble() y ChartGetDouble()

ENUM_CHART_PROPERTY_DOUBLE

Identificador	Descripción	Tipo de la propiedad
CHART_SHIFT_SIZE	Tamaño de sangría de la barra cero desde el lado derecho en	double (de 10 a 50 por ciento)

	por cientos	
CHART_FIXED_POSITION	Situación de la posición fija del gráfico desde el lado izquierdo en por cientos. La posición fija del gráfico está marcada con un pequeño triángulo gris sobre el eje horizontal de tiempo. Se muestra sólo si está desactivado el desplazamiento automático hacia la derecha con la llegada de un nuevo tick (ver la propiedad CHART_AUTOSCROLL). Cuando aumentamos o disminuimos el zoom, la barra que se encuentra en la posición fija se queda en el mismo sitio.	double
CHART_FIXED_MAX	Máximo fijo del gráfico	double
CHART_FIXED_MIN	Mínimo fijo del gráfico	double
CHART_POINTS_PER_BAR	Valor de la escala en puntos por barra	double
CHART_PRICE_MIN	Mínimo del gráfico	double r/o modificador - número de subventana
CHART_PRICE_MAX	Máximo del gráfico	double r/o modificador - número de subventana

Par la función ChartSetString() y ChartGetString()

ENUM_CHART_PROPERTY_STRING

Identificador	Descripción	Tipo de la propiedad
CHART_COMMENT	Texto del comentario en el gráfico	string

Ejemplo:

```
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);
     }
  }
```

Véase también

Ejemplos de trabajo con el gráfico

Posicionamiento de gráfico

Tres identificadores de la lista ENUM_CHART_POSITION son posibles valores del parámetro position para la función ChartNavigate().

ENUM_CHART_POSITION

Identificador	Descripción
CHART_BEGIN	Inicio del gráfico (precios más viejos)
CHART_CURRENT_POS	Posición actual
CHART_END	Fin del gráfico (precios más recientes)

Ejemplo:

```
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();
  }
```

Visualización de gráficos

Los gráficos de precios se puede representar de tres maneras:

- como barras;
- como velas;
- como línea quebrada.

El tipo concreto de mostrar el gráfico de precio se establece por la función ChartSetInteger (handle_de_gráfico,CHART_MODE, tipo_de_gráfico), donde el tipo_de_gráfico es uno de los valores de enumeración ENUM_CHART_MODE.

ENUM_CHART_MODE

Identificador	Descripción
CHART_BARS	Representación en forma de barras
CHART_CANDLES	Representación en forma de velas japonesas
CHART_LINE	Representación en forma de una línea trazada por los precios Close

Para especificar el régimen de mostrar volúmenes en el gráfico de precio utilizan la función ChartSetInteger(handle_de_gráfico, CHART_SHOW_VOLUMES, tipo_de_mostrar), donde tipo_de_mostrar es uno de los valores de enumeración ENUM_CHART_VOLUME_MODE.

ENUM_CHART_VOLUME_MODE

Identificador	Descripción
CHART_VOLUME_HIDE	Volúmenes no se muestran
CHART_VOLUME_TICK	Volúmenes de tick
CHART_VOLUME_REAL	Volúmenes comerciales

Ejemplo:

```
//--- obtenemos manejador (handle) del gráfico corriente
   long handle=ChartID();
   if(handle>0) // en caso de éxito, personalizamos más
     {
      //--- deshabilitamos desplazamiento automático
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      //--- fijamos la sangría del lado derecho del gráfico
      ChartSetInteger(handle,CHART_SHIFT,true);
      //--- representamos en forma de velas
      ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
      //--- desplazamos a 100 barras desde el inicio de historial
      ChartNavigate(handle,CHART_CURRENT_POS,100);
      //--- establecemos el modo de mostrar volúmenes de tick
      ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);
```

```
   }
```

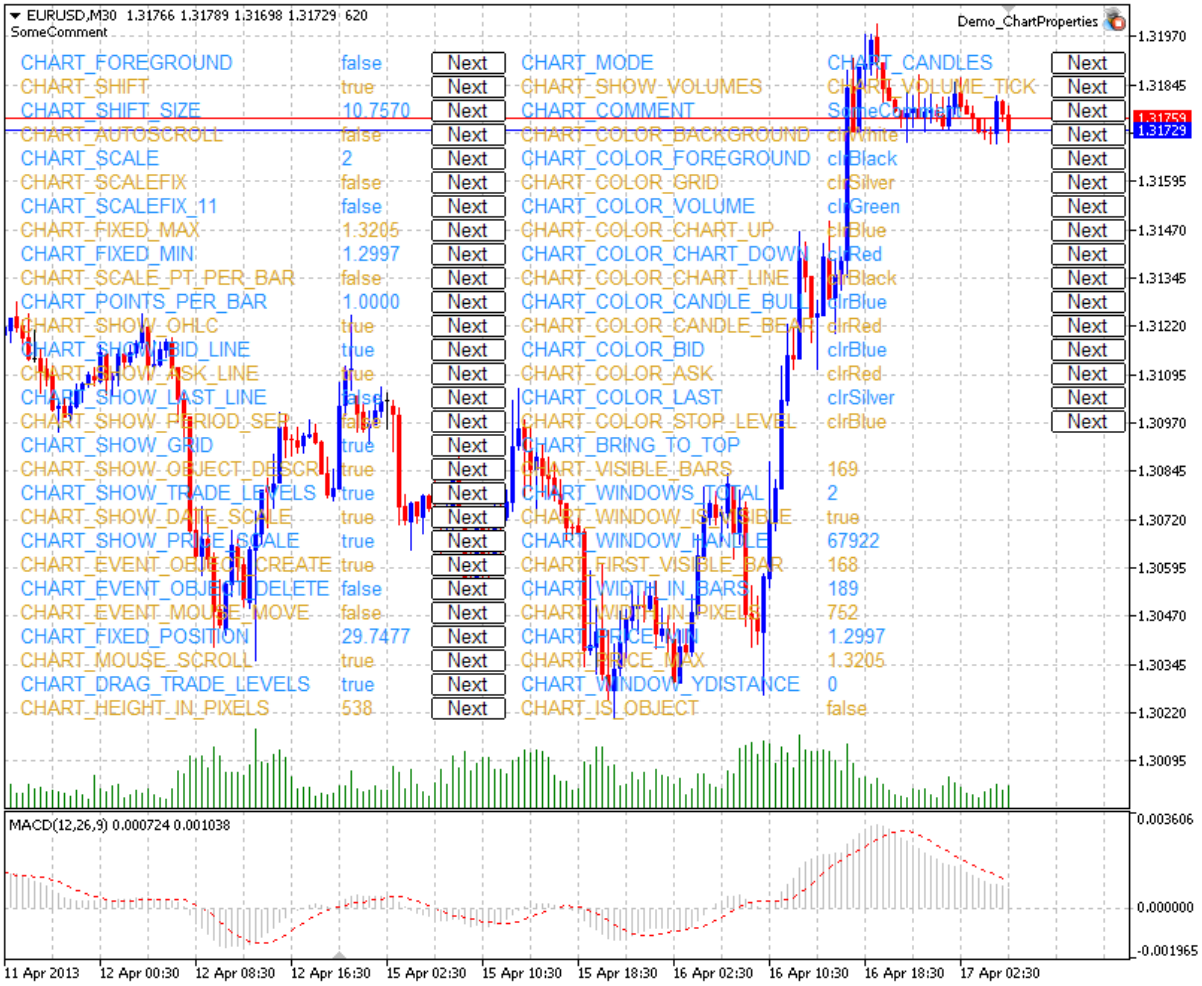
Véase también

ChartOpen, ChartID

Ejemplos de trabajo con el gráfico

Este apartado contiene varios ejemplos para el trabajo con las propiedades del gráfico. Para cada propiedad se exponen una o dos funciones terminadas que permiten establecer / obtener el valor de esta propiedad. Puede utilizar estas funciones en sus programas MQL5 tal como están.

En la imagen de abajo se ve un panel gráfico que demuestra de forma muy clara cómo el cambio de una propiedad del gráfico cambia su apariencia. El hacer click en el botón "Next" permite establecer un valor nuevo para la propiedad correspondiente y ver los cambios en la ventana del gráfico que eso provoca.



El código fuente de este panel se encuentra más abajo.

Propiedades del gráfico y ejemplos de las funciones para trabajar con ellas

- **CHART_IS_OBJECT** - determina si el objeto es un gráfico real o un objeto gráfico.

```
//+------------------------------------------------------------------+
//| Determinar que si el objeto es un gráfico. Si es                 |
//| un objeto gráfico, el resultado es true. Si es un                |
//| gráfico real, entonces la variable result obtiene el valor false.|
```

```
//+------------------------------------------------------------------+
bool ChartIsObject(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos la propiedad del gráfico
   if(!ChartGetInteger(chart_ID,CHART_IS_OBJECT,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      //--- devolvemos false
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_BRING_TO_TOP** - muestra el gráfico por encima de los demás.

```
//+------------------------------------------------------------------+
//| Enviar al terminal los comandos para mostrar el gráfico por encima de los demás.
//+------------------------------------------------------------------+
bool ChartBringToTop(const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- mostramos el gráfico por encima de los demás
   if(!ChartSetInteger(chart_ID,CHART_BRING_TO_TOP,0,true))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_MOUSE_SCROLL** - la propiedad de desplazamiento del gráfico con el botón izquierdo del ratón.

```
//+------------------------------------------------------------------+
//| La función determina si se puede desplazar el gráfico utilizando el botón izquier
```

```
//| del ratón.                                                       |
//+------------------------------------------------------------------+
bool ChartMouseScrollGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de desplazamiento del gráfico con el botón
//| del ratón.                                                       |
//+------------------------------------------------------------------+
bool ChartMouseScrollSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_MOUSE_SCROLL,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_EVENT_MOUSE_MOVE** - la propiedad del envío de los mensajes sobre los eventos de desplazamiento y pulsación de los botones del ratón a los programas mql5 (CHARTEVENT_MOUSE_MOVE).

```
//+------------------------------------------------------------------+
//| Comprobar si se envían los mensajes sobre los eventos de desplazamiento y pulsaci
//| del ratón en este gráfico a los programas mql5.                  |
//+------------------------------------------------------------------+
bool ChartEventMouseMoveGet(bool &result,const long chart_ID=0)
```

```
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo del envío de los mensajes sobre los eventos
//| de desplazamiento y pulsación de los botones del ratón a todos los programas mql5
//| gráfico.                                                         |
//+------------------------------------------------------------------+
bool ChartEventMouseMoveSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_EVENT_MOUSE_MOVE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_EVENT_OBJECT_CREATE** - la propiedad del envío de los mensajes sobre el evento de creación de un objeto gráfico a los programas mql5 (CHARTEVENT_OBJECT_CREATE).

```
//+------------------------------------------------------------------+
//| Comprobar si se envían los mensajes sobre el evento de creación del objeto gráfic
//| a todos los programas mql5 en este gráfico.                      |
//+------------------------------------------------------------------+
bool ChartEventObjectCreateGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
```

```
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo del envío de los mensajes sobre el evento
//| de creación de un objeto gráfico a todos los programas mql5 en este     |
//| gráfico.                                                         |
//+------------------------------------------------------------------+
bool ChartEventObjectCreateSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_CREATE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_EVENT_OBJECT_DELETE** - la propiedad del envío de los mensajes sobre el evento de eliminación de un objeto gráfico a los programas mql5 (CHARTEVENT_OBJECT_DELETE).

```
//+------------------------------------------------------------------+
//| Comprobar si se envían los mensajes sobre el evento de eliminación del objeto grá
//| a todos los programas mql5 en este gráfico.                      |
//+------------------------------------------------------------------+
bool ChartEventObjectDeleteGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
```

```
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo del envío de los mensajes sobre el evento
//| de eliminación de un objeto gráfico a todos los programas mql5 en este     |
//| gráfico.                                                         |
//+------------------------------------------------------------------+
bool ChartEventObjectDeleteSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_EVENT_OBJECT_DELETE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_MODE** - tipo del gráfico (velas, barras u línea).

```
//+------------------------------------------------------------------+
//| Obtener el tipo de visualización del gráfico (en forma de velas, barras o     |
//| línea).                                                          |
//+------------------------------------------------------------------+
ENUM_CHART_MODE ChartModeGet(const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=WRONG_VALUE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_MODE,0,result))
```

```
      {
       //--- mostramos el mensaje del error en el diario "Asesores Expertos"
       Print(__FUNCTION__+", Error Code = ",GetLastError());
      }
//--- devolvemos el valor de la propiedad del gráfico
   return((ENUM_CHART_MODE)result);
  }
//+------------------------------------------------------------------+
//| Establecer el tipo de visualización del gráfico (en forma de velas, barras o
//| línea).                                                          |
//+------------------------------------------------------------------+
bool ChartModeSet(const long value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_MODE,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_FOREGROUND** - la propiedad de visualización del gráfico de precios en el primer plano.

```
//+------------------------------------------------------------------+
//| La función determina si se visualiza el gráfico de precios en el primer   |
//| plano.                                                           |
//+------------------------------------------------------------------+
bool ChartForegroundGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_FOREGROUND,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
```

```
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización del gráfico de precios en
//| el primer plano.                                                 |
//+------------------------------------------------------------------+
bool ChartForegroundSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_FOREGROUND,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHIFT** - el modo de desplazamiento del gráfico de precios desde el borde derecho.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de visualización del gráfico de pre
//| con el desplazamiento desde el borde derecho.
//+------------------------------------------------------------------+
bool ChartShiftGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHIFT,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
```

```
//| La función activa / desactiva el modo de visualización del gráfico de precios con
//| el desplazamiento desde el borde derecho.
//+------------------------------------------------------------------+
bool ChartShiftSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHIFT,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_AUTOSCROLL** - el modo de desplazamiento automático hacia el borde derecho del gráfico.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de desplazamiento automático    |
//| del gráfico hacia la derecha cuando se reciben nuevos ticks.
//+------------------------------------------------------------------+
bool ChartAutoscrollGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de desplazamiento automático del gráfico|
//| hacia la derecha cuando se reciben nuevos ticks.                              |
//+------------------------------------------------------------------+
bool ChartAutoscrollSet(const bool value,const long chart_ID=0)
```

```
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_AUTOSCROLL,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SCALE** - la propiedad de la escala del gráfico.

```
//+------------------------------------------------------------------+
//| Obtener la escala del gráfico (de 0 a 5).                        |
//+------------------------------------------------------------------+
int ChartScaleGet(const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=-1;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SCALE,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((int)result);
  }
//+------------------------------------------------------------------+
//| Establecer la escala del gráfico (de 0 a 5).                     |
//+------------------------------------------------------------------+
bool ChartScaleSet(const long value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SCALE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
```

```
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SCALEFIX** - el modo de la escala fija del gráfico.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de la escala fija del gráfico.    |
//+------------------------------------------------------------------+
bool ChartScaleFixGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SCALEFIX,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de la escala fija.        |
//+------------------------------------------------------------------+
bool ChartScaleFixSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SCALEFIX,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SCALEFIX_11** - el modo de la escala del gráfico 1:1.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de la escala "1:1".|
//+------------------------------------------------------------------+
bool ChartScaleFix11Get(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de la escala "1:1"         |
//+------------------------------------------------------------------+
bool ChartScaleFix11Set(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SCALEFIX_11,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SCALE_PT_PER_BAR** - el modo de especificación de la escala del gráfico en puntos por barra.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de especificación de la escala en p
//| por barra.                                                       |
//+------------------------------------------------------------------+
```

```
bool ChartScalePerBarGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de especificación de la escala en puntos po
//| barra.                                                           |
//+------------------------------------------------------------------+
bool ChartScalePerBarSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SCALE_PT_PER_BAR,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_OHLC** - la propiedad de visualización de los valores OHLC en la esquina superior izquierda.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de visualización de los valores OHL
//| en la esquina superior izquierda.                                |
//+------------------------------------------------------------------+
bool ChartShowOHLCGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
```

```
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización de los valores OHLC en
//| la esquina superior izquierda del gráfico.                       |
//+------------------------------------------------------------------+
bool ChartShowOHLCSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_OHLC,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_BID_LINE** - la propiedad de visualización del valor Bid como una línea horizontal en el gráfico.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de visualización de la línea Bid en
//| gráfico.                                                         |
//+------------------------------------------------------------------+
bool ChartShowBidLineGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
```

```
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización de la línea Bid en el
//| gráfico.                                                         |
//+------------------------------------------------------------------+
bool ChartShowBidLineSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_BID_LINE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_ASK_LINE** - la propiedad de visualización del valor Ask como una línea horizontal en el gráfico.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de visualización de la línea Ask en
//| gráfico.                                                         |
//+------------------------------------------------------------------+
bool ChartShowAskLineGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
     {
```

```
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización de la línea Ask en el
//| gráfico.                                                         |
//+------------------------------------------------------------------+
bool ChartShowAskLineSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_ASK_LINE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_LAST_LINE** - la propiedad de visualización del valor Last como una línea horizontal en el gráfico.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de visualización de la línea para e
//| la última transacción realizada.                                 |
//+------------------------------------------------------------------+
bool ChartShowLastLineGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
```

```
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización de la línea del precio de
//| transacción realizada.                                           |
//+------------------------------------------------------------------+
bool ChartShowLastLineSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_LAST_LINE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_PERIOD_SEP** - la propiedad de visualización de los separadores verticales entre los períodos adyacentes.

```
//+------------------------------------------------------------------+
//| La función determina si está activado el modo de visualización de los separadores
//| verticales entre los períodos adyacentes.                        |
//+------------------------------------------------------------------+
bool ChartShowPeriodSeparatorGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
```

```
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización de los separadores
//| verticales entre los períodos adyacentes.                        |
//+------------------------------------------------------------------+
bool ChartShowPeriodSepapatorSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_PERIOD_SEP,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_GRID** - la propiedad de visualización de cuadrícula en el gráfico.

```
//+------------------------------------------------------------------+
//| La función determina si la cuadrícula se visualiza en el gráfico.|
//+------------------------------------------------------------------+
bool ChartShowGridGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_GRID,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva la visualización de la cuadrícula en el gráfico.
```

```
//+------------------------------------------------------------------+
bool ChartShowGridSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_GRID,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_VOLUMES** - la propiedad de visualización de volúmenes en el gráfico.

```
//+------------------------------------------------------------------+
//| La función determina si se visualizan los volúmenes en el gráfico (no        |
//| se muestran, se muestran los de ticks, se muestran los reales).      |
//+------------------------------------------------------------------+
ENUM_CHART_VOLUME_MODE ChartShowVolumesGet(const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=WRONG_VALUE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_VOLUMES,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((ENUM_CHART_VOLUME_MODE)result);
  }
//+------------------------------------------------------------------+
//| La función establece el modo de visualización de los volúmenes en el gráfico.
//+------------------------------------------------------------------+
bool ChartShowVolumesSet(const long value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_VOLUMES,value))
     {
```

```
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_OBJECT_DESCR** - la propiedad de las descripciones emergentes de los objetos gráficos.

```
//+------------------------------------------------------------------+
//| La función determina si se visualizan las descripciones emergentes |
//| de los objetos gráficos al situar el cursor sobre ellos.          |
//+------------------------------------------------------------------+
bool ChartShowObjectDescriptionGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización de las descripciones emerg
//| de los objetos gráficos al situar el cursor sobre ellos.          |
//+------------------------------------------------------------------+
bool ChartShowObjectDescriptionSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_OBJECT_DESCR,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
```

```
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_VISIBLE_BARS** - determina el número de barras en el gráfico que están disponibles para la visualización.

```
//+------------------------------------------------------------------+
//| La función obtiene el número de barras que se muestran (están visibles)|
//| en la ventana del gráfico.                                       |
//+------------------------------------------------------------------+
int ChartVisibleBars(const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=-1;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_VISIBLE_BARS,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((int)result);
  }
```

- **CHART_WINDOWS_TOTAL** - determina el número total de las ventanas del gráfico, incluyendo las subventanas de los indicadores.

```
//+------------------------------------------------------------------+
//| La función obtiene el número total de las ventanas del gráfico, incluyendo las su
//| de los indicadores.                                              |
//+------------------------------------------------------------------+
int ChartWindowsTotal(const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=-1;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_WINDOWS_TOTAL,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
```

```
//--- devolvemos el valor de la propiedad del gráfico
   return((int)result);
  }
```

- **CHART_WINDOW_IS_VISIBLE** - determina la visibilidad de la subventana.

```
//+------------------------------------------------------------------+
//| La función determina si esta ventana o subventana del gráfico es  |
//| visible.                                                          |
//+------------------------------------------------------------------+
bool ChartWindowsIsVisible(bool &result,const long chart_ID=0,const int sub_window=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_IS_VISIBLE,sub_window,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_WINDOW_HANDLE** - devuelve el manejador del gráfico.

```
//+------------------------------------------------------------------+
//| La función obtiene el manejador del gráfico                      |
//+------------------------------------------------------------------+
int ChartWindowsHandle(const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=-1;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_HANDLE,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
```

```
//--- devolvemos el valor de la propiedad del gráfico
   return((int)result);
  }
```

- **CHART_WINDOW_YDISTANCE** - determina la distancia en píxeles entre el marco superior de la subventana del indicador y el marco superior de la ventana principal del gráfico.

```
//+------------------------------------------------------------------+
//| La función obtiene la distancia en píxeles entre el marco superior de la      |
//| subventana y el marco superior de la ventana principal del gráfico.
//+------------------------------------------------------------------+
int ChartWindowsYDistance(const long chart_ID=0,const int sub_window=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=-1;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_YDISTANCE,sub_window,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((int)result);
  }
```

- **CHART_FIRST_VISIBLE_BAR** - devuelve el número de la primera barra visible en el gráfico (la indexación de las barras corresponde a la serie temporal).

```
//+------------------------------------------------------------------+
//| La función obtiene el número de la primera barra visible en el gráfico.         |
//| La indexación se realiza como en una serie temporal, las últimas barras tienen lo
//+------------------------------------------------------------------+
int ChartFirstVisibleBar(const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=-1;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_WINDOW_YDISTANCE,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
```

```
   return((int)result);
  }
```

- **CHART_WIDTH_IN_BARS** - devuelve el ancho del gráfico en barras.

```
//+------------------------------------------------------------------+
//| La función obtiene el valor del ancho del gráfico en barras.     |
//+------------------------------------------------------------------+
int ChartWidthInBars(const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=-1;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_BARS,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((int)result);
  }
```

- **CHART_WIDTH_IN_PIXELS** - devuelve el ancho del gráfico en píxeles.

```
//+------------------------------------------------------------------+
//| La función obtiene el valor del ancho del gráfico en píxeles.    |
//+------------------------------------------------------------------+
int ChartWidthInPixels(const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=-1;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_WIDTH_IN_PIXELS,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((int)result);
  }
```

- **CHART_HEIGHT_IN_PIXELS** - la propiedad del alto del gráfico en píxeles.

```
//+------------------------------------------------------------------+
//| La función obtiene el valor del alto del gráfico en píxeles.     |
//+------------------------------------------------------------------+
int ChartHeightInPixelsGet(const long chart_ID=0,const int sub_window=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long result=-1;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((int)result);
  }
//+------------------------------------------------------------------+
//| La función establece el valor del alto del gráfico en píxeles.   |
//+------------------------------------------------------------------+
bool ChartHeightInPixelsSet(const int value,const long chart_ID=0,const int sub_windo
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_HEIGHT_IN_PIXELS,sub_window,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_BACKGROUND** - color del fondo del gráfico.

```
//+------------------------------------------------------------------+
//| La función obtiene el color del fondo del gráfico.               |
//+------------------------------------------------------------------+
color ChartBackColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
```

```
   ResetLastError();
//--- obtenemos el color del fondo del gráfico
   if(!ChartGetInteger(chart_ID,CHART_COLOR_BACKGROUND,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color del fondo del gráfico.             |
//+------------------------------------------------------------------+
bool ChartBackColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color del fondo del gráfico
   if(!ChartSetInteger(chart_ID,CHART_COLOR_BACKGROUND,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_FOREGROUND** - el color de los ejes, escala y la línea OHLC.

```
//+------------------------------------------------------------------+
//| La función obtiene el color de los ejes, escala y la línea OHLC del gráfico.
//+------------------------------------------------------------------+
color ChartForeColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color de los ejes, escala y la línea OHLC del gráfico
   if(!ChartGetInteger(chart_ID,CHART_COLOR_FOREGROUND,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
```

```
  }
//+------------------------------------------------------------------+
//| La función establece el color de los ejes, escala y la línea OHLC del gráfico.
//+------------------------------------------------------------------+
bool ChartForeColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de los ejes, escala y la línea OHLC del gráfico
   if(!ChartSetInteger(chart_ID,CHART_COLOR_FOREGROUND,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_GRID** - el color de la cuadrícula del gráfico.

```
//+------------------------------------------------------------------+
//| La función obtiene el color de la cuadrícula del gráfico.
//+------------------------------------------------------------------+
color ChartGridColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color de la cuadrícula del gráfico
   if(!ChartGetInteger(chart_ID,CHART_COLOR_GRID,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color de la cuadrícula del gráfico.
//+------------------------------------------------------------------+
bool ChartGridColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de la cuadrícula del gráfico
```

```
   if(!ChartSetInteger(chart_ID,CHART_COLOR_GRID,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_VOLUME** - el color de los volúmenes y niveles de la apertura de posiciones.

```
//+------------------------------------------------------------------+
//| La función obtiene el color de visualización de los volúmenes y niveles de la ape
//| de posiciones.                                                   |
//+------------------------------------------------------------------+
color ChartVolumeColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color de volúmenes y niveles de la apertura de posiciones
   if(!ChartGetInteger(chart_ID,CHART_COLOR_VOLUME,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color de visualización de los volúmenes y niveles de la a
//| de posiciones.                                                   |
//+------------------------------------------------------------------+
bool ChartVolumeColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de volúmenes y niveles de la apertura de posiciones
   if(!ChartSetInteger(chart_ID,CHART_COLOR_VOLUME,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
```

```
   return(true);
  }
```

- **CHART_COLOR_CHART_UP** - el color de la barra arriba, sombra y borde del cuerpo de vela alcista.

```
//+------------------------------------------------------------------+
//| La función obtiene el color de la barra que va arriba, el color de la sombra y
//| del borde del cuerpo de vela alcista.                            |
//+------------------------------------------------------------------+
color ChartUpColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color de la barra arriba, sombra y borde del cuerpo de la vela alc
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_UP,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color de la barra que va arriba, el color de la sombra y
//| del borde del cuerpo de vela alcista.                            |
//+------------------------------------------------------------------+
bool ChartUpColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de la barra arriba, sombra y borde del cuerpo de la vela
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_UP,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_CHART_DOWN** - el color de la barra abajo, sombra y borde del cuerpo de vela bajista.

```
//+------------------------------------------------------------------+
```

```
//| La función obtiene el color de la barra que va abajo, el color de la sombra y
//| del borde del cuerpo de vela bajista.                                          |
//+------------------------------------------------------------------+
color ChartDownColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color de la barra abajo, sombra y borde del cuerpo de la vela baji
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_DOWN,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color de la barra que va abajo, el color de la sombra y |
//| del borde del cuerpo de vela bajista.                                          |
//+------------------------------------------------------------------+
bool ChartDownColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de la barra abajo, sombra y borde del cuerpo de la vela b
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_DOWN,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_CHART_LINE** - el color de la línea del gráfico y de las velas japonesas "Doji".

```
//+------------------------------------------------------------------+
//| La función obtiene el color de la línea del gráfico y de las velas japonesas "Doj
//+------------------------------------------------------------------+
color ChartLineColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
```

```
   ResetLastError();
//--- obtenemos el color de la línea del gráfico y de las velas japonesas "Doji"
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CHART_LINE,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color de la línea del gráfico y de las velas japonesas
//| "Doji".                                                          |
//+------------------------------------------------------------------+
bool ChartLineColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de la línea del gráfico y de las velas japonesas "Doji"
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CHART_LINE,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_CANDLE_BULL** - el color del cuerpo de la vela alcista.

```
//+------------------------------------------------------------------+
//| La función obtiene el color del cuerpo de la vela alcista.
//+------------------------------------------------------------------+
color ChartBullColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color del cuerpo de la vela alcista
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
```

```
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color del cuerpo de la vela alcista.     |
//+------------------------------------------------------------------+
bool ChartBullColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color del cuerpo de la vela alcista
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BULL,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_CANDLE_BEAR** - el color del cuerpo de la vela bajista.

```
//+------------------------------------------------------------------+
//| La función obtiene el color del cuerpo de la vela bajista.       |
//+------------------------------------------------------------------+
color ChartBearColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color del cuerpo de la vela bajista
   if(!ChartGetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color del cuerpo de la vela bajista.     |
//+------------------------------------------------------------------+
bool ChartBearColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
```

```
//--- establecemos el color del cuerpo de la vela bajista
   if(!ChartSetInteger(chart_ID,CHART_COLOR_CANDLE_BEAR,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_BID** - el color de la línea del precio Bid.

```
//+------------------------------------------------------------------+
//| La función obtiene el color de la línea del precio Bid.          |
//+------------------------------------------------------------------+
color ChartBidColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color de la línea del precio Bid
   if(!ChartGetInteger(chart_ID,CHART_COLOR_BID,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color de la línea del precio Bid.        |
//+------------------------------------------------------------------+
bool ChartBidColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de la línea del precio Bid
   if(!ChartSetInteger(chart_ID,CHART_COLOR_BID,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
```

```
  }
```

- **CHART_COLOR_ASK** - el color de la línea del precio Ask.

```
//+------------------------------------------------------------------+
//| La función obtiene el color de la línea del precio Ask.          |
//+------------------------------------------------------------------+
color ChartAskColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color de la línea del precio Ask
   if(!ChartGetInteger(chart_ID,CHART_COLOR_ASK,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color de la línea del precio Ask.        |
//+------------------------------------------------------------------+
bool ChartAskColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de la línea del precio Ask
   if(!ChartSetInteger(chart_ID,CHART_COLOR_ASK,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_LAST** - el color de la línea del precio de la última transacción realizada (Last).

```
//+------------------------------------------------------------------+
//| La función obtiene el color de la línea del precio de la última transacción reali
//+------------------------------------------------------------------+
color ChartLastColorGet(const long chart_ID=0)
```

```
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color de la línea del precio de la última transacción realizada (L
   if(!ChartGetInteger(chart_ID,CHART_COLOR_LAST,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color de la línea del precio de la última      |
//| transacción realizada.                                                 |
//+------------------------------------------------------------------+
bool ChartLastColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de la línea del precio de la última transacción realizada
   if(!ChartSetInteger(chart_ID,CHART_COLOR_LAST,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_COLOR_STOP_LEVEL** - el color de los niveles de las órdenes Stop (Stop Loss y Take Profit).

```
//+------------------------------------------------------------------+
//| La función obtiene el color de los niveles Stop Loss y Take Profit.    |
//+------------------------------------------------------------------+
color ChartStopLevelColorGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el color
   long result=clrNONE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el color de los niveles de las órdenes Stop (Stop Loss y Take Profit)
   if(!ChartGetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,0,result))
     {
```

```
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return((color)result);
  }
//+------------------------------------------------------------------+
//| La función establece el color de los niveles Stop Loss y Take Profit.     |
//+------------------------------------------------------------------+
bool ChartStopLevelColorSet(const color clr,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el color de los niveles de las órdenes Stop (Stop Loss y Take Prof
   if(!ChartSetInteger(chart_ID,CHART_COLOR_STOP_LEVEL,clr))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_TRADE_LEVELS** - la propiedad de visualización de los niveles de tradıng en el gráfico (niveles de posiciones abiertas, Stop Loss, Take Profit y órdenes pendientes).

```
//+------------------------------------------------------------------+
//| La función determina si los niveles de trading se visualizan en el gráfico.  |
//+------------------------------------------------------------------+
bool ChartShowTradeLevelsGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
```

```
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización de los niveles de trading.
//+------------------------------------------------------------------+
bool ChartShowTradeLevelsSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_TRADE_LEVELS,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_DRAG_TRADE_LEVELS** - la propiedad del permiso para arrastrar los niveles de trading por el gráfico utilizando el ratón.

```
//+------------------------------------------------------------------+
//| La función determina si se puede arrastrar los niveles de trading en    |
//| el gráfico utilizando el ratón.                                  |
//+------------------------------------------------------------------+
bool ChartDragTradeLevelsGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de arrastre de los niveles de trading |
//| en el gráfico utilizando el ratón.                               |
```

```
//+------------------------------------------------------------------+
bool ChartDragTradeLevelsSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_DRAG_TRADE_LEVELS,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_DATE_SCALE** - la propiedad de visualización de la escala de tiempo en el gráfico.

```
//+------------------------------------------------------------------+
//| La función determina si la escala de tiempo se visualiza en el gráfico.    |
//+------------------------------------------------------------------+
bool ChartShowDateScaleGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización de la escala de tiempo en
//| gráfico.                                                         |
//+------------------------------------------------------------------+
bool ChartShowDateScaleSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
```

```
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_DATE_SCALE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHOW_PRICE_SCALE** - la propiedad de visualización de la escala de precios en el gráfico.

```
//+------------------------------------------------------------------+
//| La función determina si la escala de precios se visualiza en el gráfico.       |
//+------------------------------------------------------------------+
bool ChartShowPriceScaleGet(bool &result,const long chart_ID=0)
  {
//--- preparamos la variable para obtener el valor de la propiedad
   long value;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- guardamos en la variable el valor de la propiedad del gráfico
   result=value;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función activa / desactiva el modo de visualización de la escala de precios en
//| gráfico.                                                         |
//+------------------------------------------------------------------+
bool ChartShowPriceScaleSet(const bool value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetInteger(chart_ID,CHART_SHOW_PRICE_SCALE,0,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
```

```
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_SHIFT_SIZE** - el tamaño de desplazamiento de la barra cero desde el borde derecho en por cientos.

```
//+------------------------------------------------------------------+
//| La función obtiene el tamaño de desplazamiento de la barra cero desde el borde de
//| del gráfico en por cientos (de 10% a 50%).                       |
//+------------------------------------------------------------------+
double ChartShiftSizeGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el resultado
   double result=EMPTY_VALUE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetDouble(chart_ID,CHART_SHIFT_SIZE,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return(result);
  }
//+------------------------------------------------------------------+
//| La función establece el tamaño de desplazamiento de la barra cero desde el borde
//| del gráfico en por cientos (de 10% a 50%). Para activar el modo de   |
//| desplazamiento, hay que establecer el valor de la propiedad CHART_SHIFT igual a
//| true.                                                            |
//+------------------------------------------------------------------+
bool ChartShiftSizeSet(const double value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetDouble(chart_ID,CHART_SHIFT_SIZE,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
```

```
  }
```

- **CHART_FIXED_POSITION** - la posición fija del gráfico desde el borde izquierdo en por cientos.

```
//+------------------------------------------------------------------+
//| La función obtiene la posición fija del gráfico desde      |
//| el borde izquierdo en por cientos.                                 |
//+------------------------------------------------------------------+
double ChartFixedPositionGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el resultado
   double result=EMPTY_VALUE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetDouble(chart_ID,CHART_FIXED_POSITION,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return(result);
  }
//+------------------------------------------------------------------+
//| La función establece la posición fija del gráfico desde |
//| el borde izquierdo en por cientos. Para ver la       |
//| posición fija en el gráfico, antes hay que establecer    |
//| el valor de la propiedad CHART_AUTOSCROLL igual a false.           |
//+------------------------------------------------------------------+
bool ChartFixedPositionSet(const double value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetDouble(chart_ID,CHART_FIXED_POSITION,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_FIXED_MAX** - la propiedad del máximo fijo del gráfico.

```
//+------------------------------------------------------------------+
```

```
//| La función obtiene el valor del máximo fijo del gráfico.      |
//+------------------------------------------------------------------+
double ChartFixedMaxGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el resultado
   double result=EMPTY_VALUE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetDouble(chart_ID,CHART_FIXED_MAX,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return(result);
  }
//+------------------------------------------------------------------+
//| La función establece el valor del máximo fijo del gráfico. |
//| Para poder modificar el valor de esta propiedad, hace falta  |
//| establecer antes el valor de la propiedad CHART_SCALEFIX igual a    |
//| true.                                                            |
//+------------------------------------------------------------------+
bool ChartFixedMaxSet(const double value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetDouble(chart_ID,CHART_FIXED_MAX,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_FIXED_MIN** - la propiedad del mínimo fijo del gráfico.

```
//+------------------------------------------------------------------+
//| La función obtiene el valor del mínimo fijo del gráfico.       |
//+------------------------------------------------------------------+
double ChartFixedMinGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el resultado
   double result=EMPTY_VALUE;
```

```
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetDouble(chart_ID,CHART_FIXED_MIN,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return(result);
  }
//+------------------------------------------------------------------+
//| La función establece el valor del mínimo fijo del gráfico.  |
//| Para poder modificar el valor de esta propiedad, hace falta  |
//| establecer antes el valor de la propiedad CHART_SCALEFIX igual a    |
//| true.                                                            |
//+------------------------------------------------------------------+
bool ChartFixedMinSet(const double value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetDouble(chart_ID,CHART_FIXED_MIN,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_POINTS_PER_BAR** - el valor de la escala en puntos por barra.

```
//+------------------------------------------------------------------+
//| La función obtiene el valor de la escala del gráfico en puntos por barra.     |
//+------------------------------------------------------------------+
double ChartPointsPerBarGet(const long chart_ID=0)
  {
//--- preparamos la variable para recibir el resultado
   double result=EMPTY_VALUE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetDouble(chart_ID,CHART_POINTS_PER_BAR,0,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
```

```
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return(result);
  }
//+------------------------------------------------------------------+
//| La función establece el valor de la escala del gráfico en puntos por barra.|
//| Para ver el resultado del cambio del valor de esta    |
//| propiedad, antes hay que establecer el valor de la propiedad          |
//| CHART_SCALE_PT_PER_BAR igual a true.                              |
//+------------------------------------------------------------------+
bool ChartPointsPerBarSet(const double value,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetDouble(chart_ID,CHART_POINTS_PER_BAR,value))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

- **CHART_PRICE_MIN** - devuelve el valor del mínimo del gráfico.

```
//+------------------------------------------------------------------+
//| La función obtiene el valor del mínimo del gráfico en la ventana principal o    |
//| subventana.                                                      |
//+------------------------------------------------------------------+
double ChartPriceMin(const long chart_ID=0,const int sub_window=0)
  {
//--- preparamos la variable para recibir el resultado
   double result=EMPTY_VALUE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetDouble(chart_ID,CHART_PRICE_MIN,sub_window,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return(result);
  }
```

- **CHART_PRICE_MAX** - devuelve el valor del máximo del gráfico.

```
//+------------------------------------------------------------------+
//| La función obtiene el valor del máximo del gráfico en la ventana principal o   |
//| subventana.                                                      |
//+------------------------------------------------------------------+
double ChartPriceMax(const long chart_ID=0,const int sub_window=0)
  {
//--- preparamos la variable para recibir el resultado
   double result=EMPTY_VALUE;
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetDouble(chart_ID,CHART_PRICE_MAX,sub_window,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
     }
//--- devolvemos el valor de la propiedad del gráfico
   return(result);
  }
```

- **CHART_COMMENT** - texto del comentario en el gráfico.

```
//+------------------------------------------------------------------+
//| La función obtiene el texto del comentario en la esquina superior izquierda del g
//+------------------------------------------------------------------+
bool ChartCommentGet(string &result,const long chart_ID=0)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el valor de la propiedad
   if(!ChartGetString(chart_ID,CHART_COMMENT,result))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función establece el texto del comentario en la esquina superior izquierda
//| del gráfico.                                                     |
//+------------------------------------------------------------------+
bool ChartCommentSet(const string str,const long chart_ID=0)
```

```
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- establecemos el valor de la propiedad
   if(!ChartSetString(chart_ID,CHART_COMMENT,str))
     {
      //--- mostramos el mensaje del error en el diario "Asesores Expertos"
      Print(__FUNCTION__+", Error Code = ",GetLastError());
      return(false);
     }
//--- ejecución con éxito
   return(true);
  }
```

Panel para las propiedades del gráfico

```
//--- conectamos la biblioteca de los elementos de control
#include <ChartObjects\ChartObjectsTxtControls.mqh>
//--- constantes predefinidas
#define X_PROPERTY_NAME_1    10  // coordenada X del nombre de la propiedad en la pri
#define X_PROPERTY_VALUE_1   225 // x-coordenada X del valor de la propiedad en la pr
#define X_PROPERTY_NAME_2    345 // coordenada X del nombre de la propiedad en la seg
#define X_PROPERTY_VALUE_2   550 // coordenada X del valor de la propiedad en la segu
#define X_BUTTON_1           285 // coordenada X del botón en la primera columna
#define X_BUTTON_2           700 // coordenada X del botón en la segunda columna
#define Y_PROPERTY_1         30  // coordenada Y del inicio de la primera y la segund
#define Y_PROPERTY_2         286 // coordenada Y del inicio de la tercera columna
#define Y_DISTANCE           16  // distancia por el eje Y entre las filas
#define LAST_PROPERTY_NUMBER 111 // el número de la última propiedad gráfica
//--- parámetros de entrada
input color InpFirstColor=clrDodgerBlue; // Color de las filas impares
input color InpSecondColor=clrGoldenrod; // Color de las filas pares
//--- variables y arrays
CChartObjectLabel  ExtLabelsName[];  // etiquetas para visualizar los nombres de prop
CChartObjectLabel  ExtLabelsValue[]; // etiquetas para visualizar los valores de prop
CChartObjectButton ExtButtons[];     // botones
int                ExtNumbers[];     // índices de propiedades
string             ExtNames[];       // nombres de propiedades
uchar              ExtDataTypes[];   // tipos de datos de propiedades (integer, doubl
uint               ExtGroupTypes[];  // array que almacena los datos sobre la pertene
uchar              ExtDrawTypes[];   // array que almacena los datos sobre el modo de
double             ExtMaxValue[];    // valores máximos permitidos para las propiedad
double             ExtMinValue[];    // valores mínimos permitidos para las propiedad
double             ExtStep[];        // pasos para cambiar las propiedades
int                ExtCount;         // número total de todas las propiedades
color              ExtColors[2];     // array de colores para visualizar las filas
string             ExtComments[2];   // array de comentarios (para la propiedad CHART
```

```
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- mostramos el comentario sobre el gráfico
   Comment("SomeComment");
//--- guardamos los colores en el array para conmutarse luego entre ellos
   ExtColors[0]=InpFirstColor;
   ExtColors[1]=InpSecondColor;
//--- guardamos los comentarios en el array para conmutarse luego entre ellos
   ExtComments[0]="FirstComment";
   ExtComments[1]="SecondComment";
//--- preparamos y visualizamos el panel de control de las propiedades del gráfico
   if(!PrepareControls())
      return(INIT_FAILED);
//--- ejecución con éxito
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Deinitialization function of the expert                          |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- quitamos el texto del comentario en el gráfico
   Comment("");
  }
//+------------------------------------------------------------------+
//| Manejador de eventos del gráfico                                 |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- comprobación del evento de pinchar sobre un objeto del gráfico
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      //--- dividimos el nombre del objeto por el separador
      string obj_name[];
      StringSplit(sparam,'_',obj_name);
      //--- prueba de que si el objeto es un botón
      if(obj_name[0]=="Button")
        {
         //--- obtenemos el índice del botón
         int index=(int)StringToInteger(obj_name[1]);
         //--- ponemos el botón en el estado no presionado
         ExtButtons[index].State(false);
```

```
         //--- establecemos nuevo valor de la propiedad en función de su tipo
         if(ExtDataTypes[index]=='I')
            ChangeIntegerProperty(index);
         if(ExtDataTypes[index]=='D')
            ChangeDoubleProperty(index);
         if(ExtDataTypes[index]=='S')
            ChangeStringProperty(index);
        }
     }
//--- redibujo de valores de propiedad
   RedrawProperties();
   ChartRedraw();
  }
//+------------------------------------------------------------------+
//| Cambio de la propiedad integer del gráfico                       |
//+------------------------------------------------------------------+
void ChangeIntegerProperty(const int index)
  {
//--- obtenemos el valor actual de la propiedad
   long value=ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index]);
//--- determinamos el siguiente valor de la propiedad
   switch(ExtDrawTypes[index])
     {
      case 'C':
         value=GetNextColor((color)value);
         break;
      default:
         value=(long)GetNextValue((double)value,index);
         break;
     }
//--- establecemos nuevo valor de la propiedad
   ChartSetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[index],0,value);
  }
//+------------------------------------------------------------------+
//| Cambio de la propiedad double del gráfico                        |
//+------------------------------------------------------------------+
void ChangeDoubleProperty(const int index)
  {
//--- obtenemos el valor actual de la propiedad
   double value=ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index]);
//--- determinamos el siguiente valor de la propiedad
   value=GetNextValue(value,index);
//--- establecemos nuevo valor de la propiedad
   ChartSetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbers[index],value);
  }
//+------------------------------------------------------------------+
//| Cambio de la propiedad string del gráfico                        |
//+------------------------------------------------------------------+
```

```
void ChangeStringProperty(const int index)
  {
//--- variable estática para conmutar dentro del array de comentarios ExtComments
   static uint comment_index=1;
//--- cambiamos el índice para obtener otro comentario
   comment_index=1-comment_index;
//--- establecemos nuevo valor de la propiedad
   ChartSetString(0,(ENUM_CHART_PROPERTY_STRING)ExtNumbers[index],ExtComments[comment
  }
//+------------------------------------------------------------------+
//| Determinación del siguiente valor de la propiedad                |
//+------------------------------------------------------------------+
double GetNextValue(const double value,const int index)
  {
   if(value+ExtStep[index]<=ExtMaxValue[index])
      return(value+ExtStep[index]);
   else
      return(ExtMinValue[index]);
  }
//+------------------------------------------------------------------+
//| Obtención del siguiente color para la propiedad del tipo color   |
//+------------------------------------------------------------------+
color GetNextColor(const color clr)
  {
//--- devolvemos el siguiente valor del color
   switch(clr)
     {
      case clrWhite: return(clrRed);
      case clrRed:   return(clrGreen);
      case clrGreen: return(clrBlue);
      case clrBlue:  return(clrBlack);
      default:       return(clrWhite);
     }
  }
//+------------------------------------------------------------------+
//| Redibujo de valores de las propiedades                           |
//+------------------------------------------------------------------+
void RedrawProperties(void)
  {
//--- texto del valor de la propiedad
   string text;
   long   value;
//--- ciclo según el número de propiedades
   for(int i=0;i<ExtCount;i++)
     {
      text="";
      switch(ExtDataTypes[i])
        {
```

```
         case 'I':
            //--- obtenemos el valor actual de la propiedad
            if(!ChartGetInteger(0,(ENUM_CHART_PROPERTY_INTEGER)ExtNumbers[i],0,value)
            break;
            //--- texto de la propiedad integer
            switch(ExtDrawTypes[i])
              {
               //--- propiedad del color
               case 'C':
                  text=(string)((color)value);
                  break;
                  //--- propiedad booleana
               case 'B':
                  text=(string)((bool)value);
                  break;
                  //--- propiedad de la enumeración ENUM_CHART_MODE
               case 'M':
                  text=EnumToString((ENUM_CHART_MODE)value);
                  break;
                  //--- propiedad de la enumeración ENUM_CHART_VOLUME_MODE
               case 'V':
                  text=EnumToString((ENUM_CHART_VOLUME_MODE)value);
                  break;
                  //--- número del tipo int
               default:
                  text=IntegerToString(value);
                  break;
              }
            break;
         case 'D':
            //--- texto de la propiedad double
            text=DoubleToString(ChartGetDouble(0,(ENUM_CHART_PROPERTY_DOUBLE)ExtNumbe
            break;
         case 'S':
            //--- texto de la propiedad string
            text=ChartGetString(0,(ENUM_CHART_PROPERTY_STRING)ExtNumbers[i]);
            break;
        }
      //--- visualizamos el valor de la propiedad
      ExtLabelsValue[i].Description(text);
     }
  }
//+------------------------------------------------------------------+
//| Crear el panel para gestionar las propiedades del gráfico        |
//+------------------------------------------------------------------+
bool PrepareControls(void)
  {
//--- adjudicamos la memoria para los arrays con reserva
```

```
   MemoryAllocation(LAST_PROPERTY_NUMBER+1);
//--- variables
   int i=0;      // variable del ciclo
   int col_1=0; // número de propiedades en la primera columna
   int col_2=0; // número de propiedades en la segunda columna
   int col_3=0; // número de propiedades en la tercera columna
//--- número de propiedades actuales - 0
   ExtCount=0;
//--- buscamos las propiedades en el ciclo
   while(i<=LAST_PROPERTY_NUMBER)
     {
      //--- guardamos el número actual de la propiedad
      ExtNumbers[ExtCount]=i;
      //--- aumentamos el valor de la variable del ciclo
      i++;
      //--- comprobamos si hay propiedad con este número
      if(CheckNumber(ExtNumbers[ExtCount],ExtNames[ExtCount],ExtDataTypes[ExtCount],E
        {
         //--- creamos los elementos de control para la propiedad
         switch(ExtGroupTypes[ExtCount])
           {
            case 1:
               //--- creamos las etiquetas y el botón para la propiedad
               if(!ShowProperty(ExtCount,0,X_PROPERTY_NAME_1,X_PROPERTY_VALUE_1,X_BUT
               return(false);
               //--- el número de elementos en la primera columna se ha aumentado
               col_1++;
               break;
            case 2:
               //--- creamos las etiquetas y el botón para la propiedad
               if(!ShowProperty(ExtCount,1,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,X_BUT
               return(false);
               //--- el número de elementos en la segunda columna se ha aumentado
               col_2++;
               break;
            case 3:
               //--- creamos sólo las etiquetas para la propiedad
               if(!ShowProperty(ExtCount,2,X_PROPERTY_NAME_2,X_PROPERTY_VALUE_2,0,Y_P
               return(false);
               //--- el número de elementos en la tercera columna se ha aumentado
               col_3++;
               break;
           }
         //--- determinamos el valor máximo, mínimo y el paso
         GetMaxMinStep(ExtNumbers[ExtCount],ExtMaxValue[ExtCount],ExtMinValue[ExtCoun
         //--- aumentamos el número de propiedades
         ExtCount++;
        }
```

```
      }
//--- liberamos la memoria que no se utiliza por los arrays
   MemoryAllocation(ExtCount);
//--- redibujamos los valores de las propiedades
   RedrawProperties();
   ChartRedraw();
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| Adjudicación de memoria para los 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);
  }
//+------------------------------------------------------------------+
//| Comprobamos si el índice de la propiedad pertenece a una de      |
//| las enumeraciones ENUM_CHART_PROPERTIES                          |
//+------------------------------------------------------------------+
bool CheckNumber(const int ind,string &name,uchar &data_type,uint &group_type,uchar &
  {
//--- comprobamos si la propiedad es del tipo entero (integer)
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_INTEGER)ind);
   if(_LastError==0)
     {
      data_type='I';                        // propiedad de la enumeración ENUM_CHART_P
      GetTypes(ind,group_type,draw_type); // definimos los parámetros de visualizació
      return(true);
     }
//--- comprobamos si la propiedad es del tipo real (double)
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_DOUBLE)ind);
   if(_LastError==0)
     {
      data_type='D';                        // propiedad de la enumeración ENUM_CHART_P
      GetTypes(ind,group_type,draw_type); // definimos los parámetros de visualizació
```

```
      return(true);
     }
//--- comprobamos si la propiedad es del tipo literal (string)
   ResetLastError();
   name=EnumToString((ENUM_CHART_PROPERTY_STRING)ind);
   if(_LastError==0)
     {
      data_type='S';                        // propiedad de la enumeración ENUM_CHART_P
      GetTypes(ind,group_type,draw_type); // definimos los parámetros de visualizació
      return(true);
     }
//--- la propiedad no pertenece a ninguna de las enumeraciones
   return(false);
  }
//+------------------------------------------------------------------+
//| Determinación del grupo en el que debe almacenarse la propiedad,     |
//| así como su tipo de visualización                                    |
//+------------------------------------------------------------------+
void GetTypes(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- comprobamos si la propiedad pertenece al tercer grupo
//--- las propiedades del tercer grupo se muestran en la segunda columna empezando de
   if(CheckThirdGroup(property_number,group_type,draw_type))
      return;
//--- comprobamos si la propiedad pertenece al segundo grupo
//--- las propiedades del tercer grupo se muestran en la segunda columna desde su ini
   if(CheckSecondGroup(property_number,group_type,draw_type))
      return;
//--- si nos hemos encontrado aquí, entonces la propiedad pertenece al primer grupo (
   CheckFirstGroup(property_number,group_type,draw_type);
  }
//+------------------------------------------------------------------+
//| La función comprueba si la propiedad pertenece al tercer grupo, y     |
//| si es así, determina su tipo de visualización                        |
//+------------------------------------------------------------------+
bool CheckThirdGroup(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- comprobamos si la propiedad pertenece al tercer grupo
   switch(property_number)
     {
      //--- propiedades booleanas
      case CHART_IS_OBJECT:
      case CHART_WINDOW_IS_VISIBLE:
         draw_type='B';
         break;
         //--- propiedades enteras (integer)
      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;
         //--- propiedades reales (double)
      case CHART_PRICE_MIN:
      case CHART_PRICE_MAX:
         draw_type='D';
         break;
         //--- de hecho, esta propiedad es un comando de visualización del gráfico po
         //--- no hay necesidad de aplicarla para este panel, porque la ventana siemp
         //--- va a colocarse por encima de todas las demás antes de que la utilicemo
      case CHART_BRING_TO_TOP:
         draw_type=' ';
         break;
         //--- la propiedad no pertenece al tercer grupo
      default:
         return(false);
     }
//--- la propiedad pertenece al tercer grupo
   group_type=3;
   return(true);
  }
//+------------------------------------------------------------------+
//| La función comprueba si la propiedad pertenece al segundo grupo, y si |
//| es así, determina su tipo de visualización                       |
//+------------------------------------------------------------------+
bool CheckSecondGroup(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- comprobamos si la propiedad pertenece al segundo grupo
   switch(property_number)
     {
      //--- la propiedad del tipo ENUM_CHART_MODE
      case CHART_MODE:
         draw_type='M';
         break;
         //--- la propiedad del tipo ENUM_CHART_VOLUME_MODE
      case CHART_SHOW_VOLUMES:
         draw_type='V';
         break;
         //--- propiedad literal (string)
      case CHART_COMMENT:
         draw_type='S';
         break;
         //--- propiedad del color
```

```
      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;
         //--- la propiedad no pertenece al segundo grupo
      default:
         return(false);
     }
//--- la propiedad pertenece al segundo grupo
   group_type=2;
   return(true);
  }
//+------------------------------------------------------------------+
//| Esta función se invoca sólo si ya se sabe   |
//| que la propiedad no pertenece ni al segundo, ni al tercer grupo de propiedades
//+------------------------------------------------------------------+
void CheckFirstGroup(const int property_number,uint &group_type,uchar &draw_type)
  {
//--- la propiedad pertenece al primer grupo
   group_type=1;
//--- determinamos el tipo de visualización de la propiedad
   switch(property_number)
     {
      //--- propiedades enteras (integer)
      case CHART_SCALE:
      case CHART_HEIGHT_IN_PIXELS:
         draw_type='I';
         return;
         //--- propiedades reales (double)
      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;
         //--- sólo quedan las propiedades booleanas
```

```
      default:
         draw_type='B';
         return;
      }
  }
//+------------------------------------------------------------------+
//| Crear las etiquetas y el botón para la propiedad                 |
//+------------------------------------------------------------------+
bool ShowProperty(const int ind,const int type,const int x1,const int x2,
                  const int xb,const int y,const bool btn)
  {
//--- array estático para conmutar dentro del array de color ExtColors
   static uint color_index[3]={1,1,1};
//--- cambiamos el índice para obtener otro color
   color_index[type]=1-color_index[type];
//--- mostramos etiquetas y el botón (si btn=true) para la propiedad
   if(!LabelCreate(ExtLabelsName[ind],"name_"+(string)ind,ExtNames[ind],ExtColors[col
      return(false);
   if(!LabelCreate(ExtLabelsValue[ind],"value_"+(string)ind,"",ExtColors[color_index[
      return(false);
   if(btn && !ButtonCreate(ExtButtons[ind],(string)ind,xb,y+1))
      return(false);
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| Crear etiqueta                                                   |
//+------------------------------------------------------------------+
bool LabelCreate(CChartObjectLabel &lbl,const string name,const string text,
                 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);
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| Crear el botón                                                   |
//+------------------------------------------------------------------+
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);
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| Establecemos el valor máximo, mínimo y el paso  |
//+------------------------------------------------------------------+
void GetMaxMinStep(const int property_number,double &max,double &min,double &step)
  {
   double value;
//--- establecemos valores en función del tipo de la propiedad
   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;
         //--- valores por defecto
      default:
         max=1;
         min=0;
         step=1;
     }
  }
```

Constantes de objetos

Hay 39 objetos gráficos que se puede crear y mostrar en el gráfico de precios. Todas las constantes destinadas para trabajar con objetos están divididas en 9 grupos:

- Tipos de objetos – identificadores de objetos gráficos;
- Propiedades de objetos – trabajo con las propiedades de objetos gráficos;
- Modos de enlace de objetos – constantes de posicionamiento de objetos en el gráfico;
- Esquina de enlace – permite fijar la esquina del gráfico respecto a la cual se realiza el posicionamiento del objeto en píxeles;
- Visibilidad de objetos – establecimiento de períodos en los que un objeto está visible;
- Niveles de las ondas de Elliot – gradación de la marcación ondulada;
- Objetos de Gann – constantes de tendencia para el abanico de Gann y la retícula de Gann;
- Colores Web – constantes de los colores Web predefinidos;
- Wingdings – códigos de caracteres de la fuente Wingdings.

Tipos de objetos

Cuando la función ObjectCreate() crea un objeto gráfico, es necesario especificar el tipo del objeto que se va a crear, el cual podrá adquirir uno de los valores de la enumeración ENUM_OBJECT. Las siguientes especificaciones de las propiedades del objeto creado son posibles mediante las funciones para el trabajo con objetos gráficos.

ENUM_OBJECT

Identificador		Descripción
OBJ_VLINE		Línea vertical
OBJ_HLINE		Línea horizontal
OBJ_TREND		Línea de tendencia
OBJ_TRENDBYANGLE		Línea de tendencia por ángulo
OBJ_CYCLES		Líneas cíclicas
OBJ_ARROWED_LINE		Objeto "Línea de separación con una flecha"
OBJ_CHANNEL		Canal equidistante
OBJ_STDDEVCHANNEL		Canal de desviación estándar
OBJ_REGRESSION		Canal en regresión lineal
OBJ_PITCHFORK		Tridente de Andrews
OBJ_GANNLINE		Líneas de Gann
OBJ_GANNFAN		Abanico de Gann
OBJ_GANNGRID		Retícula de Gann
OBJ_FIBO		Retrocesos de Fibonacci
OBJ_FIBOTIMES		Zonas temporales de Fibonacci
OBJ_FIBOFAN		Abanico de Fibonacci
OBJ_FIBOARC		Arcos de Fibonacci
OBJ_FIBOCHANNEL		Canal de Fibonacci
OBJ_EXPANSION		Expansión de Fibonacci
OBJ_ELLIOTWAVE5		Ondas de Elliott de fase impulsiva (5)
OBJ_ELLIOTWAVE3		Ondas de Elliott de fase correctiva (3)
OBJ_RECTANGLE		Rectángulo
OBJ_TRIANGLE		Triángulo

OBJ_ELLIPSE		Elipse
OBJ_ARROW_THUMB_UP		Signo "Bien" (dedo pulgar arriba)
OBJ_ARROW_THUMB_DOWN		Signo "Mal" (dedo pulgar abajo)
OBJ_ARROW_UP		Signo "Flecha arriba"
OBJ_ARROW_DOWN		Signo "Flecha abajo"
OBJ_ARROW_STOP		Signo "Stop"
OBJ_ARROW_CHECK		Signo "Marca" (tic)
OBJ_ARROW_LEFT_PRICE		Etiqueta izquierda de precio
OBJ_ARROW_RIGHT_PRICE		Etiqueta derecha de precio
OBJ_ARROW_BUY		Signo "Buy"
OBJ_ARROW_SELL		Signo "Sell"
OBJ_ARROW		Objeto "Flecha"
OBJ_TEXT		Objeto "Texto"
OBJ_LABEL		Objeto "Etiqueta de texto"
OBJ_BUTTON		Objeto "Botón"
OBJ_CHART		Objeto "Gráfico"
OBJ_BITMAP		Objeto "Dibujo"
OBJ_BITMAP_LABEL		Objeto "Etiqueta gráfica"
OBJ_EDIT		Objeto "Campo de edición"
OBJ_EVENT		Objeto "Evento" corresponde a un evento en el calendario económico
OBJ_RECTANGLE_LABEL		Objeto "Etiqueta rectangular" sirve para crear y formatear la interfaz gráfica personalizada.

OBJ_VLINE

Вертикальная линия.

Примечание

При создании вертикальной линии, можно указать режим отображения линии на все окна графика (свойство OBJPROP_RAY).

Пример

Следующий скрипт создает и перемещает на графике вертикальную линию. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Вертикальная линия\"."
#property description "Дата точки привязки задается в процентах от ширины"
#property description "окна графика в барах."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="VLine";      // Имя линии
input int              InpDate=25;           // Дата линии в %
input color            InpColor=clrRed;      // Цвет линии
input ENUM_LINE_STYLE  InpStyle=STYLE_DASH;  // Стиль линии
input int              InpWidth=3;           // Толщина линии
input bool             InpBack=false;        // Линия на заднем плане
input bool             InpSelection=true;    // Выделить для перемещений
input bool             InpRay=true;          // Продолжение линии вниз
```

```
input bool            InpHidden=true;      // Скрыт в списке объектов
input long            InpZOrder=0;         // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает вертикальную линию                                       |
//+------------------------------------------------------------------+
bool VLineCreate(const long            chart_ID=0,        // ID графика
                 const string          name="VLine",      // имя линии
                 const int             sub_window=0,      // номер подокна
                 datetime              time=0,            // время линии
                 const color           clr=clrRed,        // цвет линии
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                 const int             width=1,           // толщина линии
                 const bool            back=false,        // на заднем плане
                 const bool            selection=true,    // выделить для перемещений
                 const bool            ray=true,          // продолжение линии вниз
                 const bool            hidden=true,       // скрыт в списке объектов
                 const long            z_order=0)         // приоритет на нажатие мыш
  {
//--- если время линии не задано, то проводим ее через последний бар
   if(!time)
      time=TimeCurrent();
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим вертикальную линию
   if(!ObjectCreate(chart_ID,name,OBJ_VLINE,sub_window,time,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать вертикальную линию! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет линии
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим отображения линии в подокнах графика
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY,ray);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
```

```
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещение вертикальной линии                                   |
//+------------------------------------------------------------------+
bool VLineMove(const long   chart_ID=0,   // ID графика
               const string name="VLine", // имя линии
               datetime     time=0)       // время линии
  {
//--- если время линии не задано, то перемещаем ее на последний бар
   if(!time)
      time=TimeCurrent();
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим вертикальную линию
   if(!ObjectMove(chart_ID,name,0,time,0))
     {
      Print(__FUNCTION__,
            ": не удалось переместить вертикальную линию! Код ошибки = ",GetLastError
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет вертикальную линию                                       |
//+------------------------------------------------------------------+
bool VLineDelete(const long   chart_ID=0,   // ID графика
                 const string name="VLine") // имя линии
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим вертикальную линию
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить вертикальную линию! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
```

```
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- массив для хранения значений дат, которые будут использованы
//--- для установки и изменения координаты точки привязки линии
   datetime date[];
//--- выделение памяти
   ArrayResize(date,bars);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- определим точки для рисования линии
   int d=InpDate*(bars-1)/100;
//--- создадим вертикальную линию
   if(!VLineCreate(0,InpName,0,date[d],InpColor,InpStyle,InpWidth,InpBack,
      InpSelection,InpRay,InpHidden,InpZOrder))
      return;
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать линию
//--- счетчик цикла
   int h_steps=bars/2;
//--- перемещаем линию
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d<bars-1)
         d+=1;
      //--- сдвигаем точку
      if(!VLineMove(0,InpName,date[d]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
```

```
      // задержка в 0.03 секунды
      Sleep(30);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим канал с графика
   VLineDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_HLINE

Горизонтальная линия.

Пример

Следующий скрипт создает и перемещает на графике горизонтальную линию. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Горизонтальная линия\"."
#property description "Цена точки привязки задается в процентах от высоты"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="HLine";      // Имя линии
input int              InpPrice=25;          // Цена линии в %
input color            InpColor=clrRed;      // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линии
input int              InpWidth=3;           // Толщина линии
input bool             InpBack=false;        // Линия на заднем плане
input bool             InpSelection=true;    // Выделить для перемещений
input bool             InpHidden=true;       // Скрыт в списке объектов
input long             InpZOrder=0;          // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает горизонтальную линию                                     |
```

```
//+------------------------------------------------------------------+
bool HLineCreate(const long            chart_ID=0,        // ID графика
                 const string          name="HLine",      // имя линии
                 const int             sub_window=0,      // номер подокна
                 double                price=0,           // цена линии
                 const color           clr=clrRed,        // цвет линии
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                 const int             width=1,           // толщина линии
                 const bool            back=false,        // на заднем плане
                 const bool            selection=true,    // выделить для перемещений
                 const bool            hidden=true,       // скрыт в списке объектов
                 const long            z_order=0)         // приоритет на нажатие мыш
  {
//--- если цена не задана, то установим ее на уровне текущей цены Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим горизонтальную линию
   if(!ObjectCreate(chart_ID,name,OBJ_HLINE,sub_window,0,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать горизонтальную линию! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет линии
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещение горизонтальной линии                                 |
```

```
//+------------------------------------------------------------------+
bool HLineMove(const long   chart_ID=0,   // ID графика
               const string name="HLine", // имя линии
               double       price=0)      // цена линии
  {
//--- если цена линии не задана, то перемещаем ее на уровень текущей цены Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим горизонтальную линию
   if(!ObjectMove(chart_ID,name,0,0,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить горизонтальную линию! Код ошибки = ",GetLastErr
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет горизонтальную линию                                     |
//+------------------------------------------------------------------+
bool HLineDelete(const long   chart_ID=0,   // ID графика
                 const string name="HLine") // имя линии
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим горизонтальную линию
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить горизонтальную линию! Код ошибки = ",GetLastError()
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
```

```
     }
//--- размер массива price
   int accuracy=1000;
//--- массив для хранения значений цен, которые будут использованы
//--- для установки и изменения координаты точки привязки линии
   double price[];
//--- выделение памяти
   ArrayResize(price,accuracy);
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования линии
   int p=InpPrice*(accuracy-1)/100;
//--- создадим горизонтальную линию
   if(!HLineCreate(0,InpName,0,price[p],InpColor,InpStyle,InpWidth,InpBack,
      InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать линию
//--- счетчик цикла
   int v_steps=accuracy/2;
//--- перемещаем линию
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p<accuracy-1)
         p+=1;
      //--- сдвигаем точку
      if(!HLineMove(0,InpName,price[p]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим с графика
```

```
   HLineDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_TREND

Трендовая линия.

Примечание

Для трендовой линии можно указать режим продолжения ее отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно).

Пример

Следующий скрипт создает и перемещает на графике трендовую линию. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Трендовая линия\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="Trend";      // Имя линии
input int             InpDate1=35;          // Дата 1-ой точки в %
input int             InpPrice1=60;         // Цена 1-ой точки в %
input int             InpDate2=65;          // Дата 2-ой точки в %
input int             InpPrice2=40;         // Цена 2-ой точки в %
input color           InpColor=clrRed;      // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линии
```

```
input int              InpWidth=2;          // Толщина линии
input bool             InpBack=false;       // Линия на заднем плане
input bool             InpSelection=true;   // Выделить для перемещений
input bool             InpRayLeft=false;    // Продолжение линии влево
input bool             InpRayRight=false;   // Продолжение линии вправо
input bool             InpHidden=true;      // Скрыт в списке объектов
input long             InpZOrder=0;         // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает линию тренда по заданным координатам                     |
//+------------------------------------------------------------------+
bool TrendCreate(const long            chart_ID=0,        // ID графика
                 const string          name="TrendLine",  // имя линии
                 const int             sub_window=0,      // номер подокна
                 datetime              time1=0,           // время первой точки
                 double                price1=0,          // цена первой точки
                 datetime              time2=0,           // время второй точки
                 double                price2=0,          // цена второй точки
                 const color           clr=clrRed,        // цвет линии
                 const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                 const int             width=1,           // толщина линии
                 const bool            back=false,        // на заднем плане
                 const bool            selection=true,    // выделить для перемещений
                 const bool            ray_left=false,    // продолжение линии влево
                 const bool            ray_right=false,   // продолжение линии вправо
                 const bool            hidden=true,       // скрыт в списке объектов
                 const long            z_order=0)         // приоритет на нажатие мыш
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeTrendEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим трендовую линию по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_TREND,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": не удалось создать линию тренда! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет линии
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
```

```
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения линии влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения линии вправо
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки линии тренда                           |
//+------------------------------------------------------------------+
bool TrendPointChange(const long   chart_ID=0,       // ID графика
                      const string name="TrendLine", // имя линии
                      const int    point_index=0,    // номер точки привязки
                      datetime     time=0,           // координата времени точки прив
                      double       price=0)          // координата цены точки привязк
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки линии тренда
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Функция удаляет линию тренда с графика.                          |
//+------------------------------------------------------------------+
bool TrendDelete(const long   chart_ID=0,       // ID графика
                 const string name="TrendLine") // имя линии
  {
//--- сбросим значение ошибки
```

```
   ResetLastError();
//--- удалим линию тренда
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить линию тренда! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки линии тренда, и для пустых     |
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
  {
//--- если время первой точки не задано, то она будет на текущем баре
   if(!time1)
      time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее второй
   if(!time2)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- установим вторую точку на 9 баров левее первой
      time2=temp[0];
     }
//--- если цена второй точки не задана, то она совпадает с ценой первой точки
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
```

```
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки линии
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования линии
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- создадим линию тренда
   if(!TrendCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,InpStyl
      InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки линии
//--- счетчик цикла
   int v_steps=accuracy/5;
//--- перемещаем первую точку привязки по вертикали
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
```

```
      if(p1>1)
         p1-=1;
      //--- сдвигаем точку
      if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- перемещаем вторую точку привязки по вертикали
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p2<accuracy-1)
         p2+=1;
      //--- сдвигаем точку
      if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в пол секунды
   Sleep(500);
//--- счетчик цикла
   int h_steps=bars/2;
//--- перемещаем обе точки привязки по горизонтали одновременно
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующие значения
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- сдвигаем точки
      if(!TrendPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!TrendPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
```

```
      // задержка в 0.03 секунды
      Sleep(30);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим трендовую линию
   TrendDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_TRENDBYANGLE

Трендовая линия по углу.

Примечание

Для трендовой линии можно указать режим продолжения ее отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно).

Для установки наклона линии можно использовать как угол, так и координаты второй точки привязки.

Пример

Следующий скрипт создает и перемещает на графике трендовую линию. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Трендовая линия по углу\"."
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="Trend";      // Имя линии
input int              InpDate1=50;          // Дата 1-ой точки в %
input int              InpPrice1=75;         // Цена 1-ой точки в %
input int              InpAngle=0;           // Угол наклона линии
input color            InpColor=clrRed;      // Цвет линии
```

```
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линии
input int             InpWidth=2;          // Толщина линии
input bool            InpBack=false;       // Линия на заднем плане
input bool            InpSelection=true;   // Выделить для перемещений
input bool            InpRayLeft=false;    // Продолжение линии влево
input bool            InpRayRight=true;    // Продолжение линии вправо
input bool            InpHidden=true;      // Скрыт в списке объектов
input long            InpZOrder=0;         // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает линию тренда по углу                                     |
//+------------------------------------------------------------------+
bool TrendByAngleCreate(const long            chart_ID=0,        // ID графика
                        const string          name="TrendLine",  // имя линии
                        const int             sub_window=0,      // номер подокна
                        datetime              time=0,            // время точки
                        double                price=0,           // цена точки
                        const double          angle=45.0,        // угол наклона
                        const color           clr=clrRed,        // цвет линии
                        const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                        const int             width=1,           // толщина линии
                        const bool            back=false,        // на заднем плане
                        const bool            selection=true,    // выделить для пере
                        const bool            ray_left=false,    // продолжение линии
                        const bool            ray_right=true,    // продолжение линии
                        const bool            hidden=true,       // скрыт в списке об
                        const long            z_order=0)         // приоритет на нажа
  {
//--- для того, чтобы было удобно перемещать трендовую линию мышью, создадим вторую т
   datetime time2=0;
   double   price2=0;
//--- установим координаты точек привязки, если они не заданы
   ChangeTrendEmptyPoints(time,price,time2,price2);
//--- сбросим значение ошибки
   ResetLastError();
//--- строим трендовую линию по 2-ум точкам
   if(!ObjectCreate(chart_ID,name,OBJ_TRENDBYANGLE,sub_window,time,price,time2,price2
     {
      Print(__FUNCTION__,
            ": не удалось создать линию тренда! Код ошибки = ",GetLastError());
      return(false);
     }
//--- изменяем угол наклона трендовой линии; в процессе изменения угла, координата вт
//--- точки линии переопределится автоматически в соответствии с новым значением угла
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- установим цвет линии
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
```

```
//--- установим толщину линии
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения линии влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения линии вправо
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет координаты точки привязки линии тренда                  |
//+------------------------------------------------------------------+
bool TrendPointChange(const long   chart_ID=0,       // ID графика
                      const string name="TrendLine", // имя линии
                      datetime     time=0,           // координата времени точки прив
                      double       price=0)          // координата цены точки привязк
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки линии тренда
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет угол наклона линии тренда                               |
```

```
//+------------------------------------------------------------------+
bool TrendAngleChange(const long   chart_ID=0,       // ID графика
                      const string name="TrendLine", // имя линии тренда
                      const double angle=45)         // угол наклона линии тренда
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим угол наклона линии тренда
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
     {
      Print(__FUNCTION__,
            ": не удалось изменить угол наклона линии! Код ошибки = ",GetLastError())
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет линию тренда                                             |
//+------------------------------------------------------------------+
bool TrendDelete(const long   chart_ID=0,       // ID графика
                 const string name="TrendLine") // имя линии
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим линию тренда
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить линию тренда! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки линии тренда, и для пустых     |
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeTrendEmptyPoints(datetime &time1,double &price1,
                            datetime &time2,double &price2)
  {
//--- если время первой точки не задано, то она будет на текущем баре
   if(!time1)
      time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
```

```
//--- установим координаты второй, вспомогательной точки
//--- вторая точка будет лежать левее на 9 баров, и иметь ту же цену
   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()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки линии
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования линии
   int d1=InpDate1*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- создадим линию тренда
```

```
   if(!TrendByAngleCreate(0,InpName,0,date[d1],price[p1],InpAngle,InpColor,InpStyle,
      InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать и вращать линию
//--- счетчик цикла
   int v_steps=accuracy/2;
//--- перемещаем точку привязки и изменяем угол наклона линии
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1>1)
         p1-=1;
      //--- сдвигаем точку
      if(!TrendPointChange(0,InpName,date[d1],price[p1]))
         return;
      if(!TrendAngleChange(0,InpName,18*(i+1)))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим с графика
   TrendDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_CYCLES

Циклические линии.

Примечание

Расстояние между линиями задается координатами времени двух точек привязки объекта.

Пример

Следующий скрипт создает и перемещает на графике циклические линии. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит циклические линии на графике."
#property description "Координаты точек привязки задаются в процентах"
#property description "от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="Cycles";   // Имя объекта
input int              InpDate1=10;        // Дата 1-ой точки в %
input int              InpPrice1=45;       // Цена 1-ой точки в %
input int              InpDate2=20;        // Дата 2-ой точки в %
input int              InpPrice2=55;       // Цена 2-ой точки в %
input color            InpColor=clrRed;    // Цвет циклических линий
input ENUM_LINE_STYLE  InpStyle=STYLE_DOT; // Стиль циклических линий
input int              InpWidth=1;         // Толщина циклических линий
```

```
input bool             InpBack=false;       // Объект на заднем плане
input bool             InpSelection=true;   // Выделить для перемещений
input bool             InpHidden=true;      // Скрыт в списке объектов
input long             InpZOrder=0;         // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает циклические линии                                        |
//+------------------------------------------------------------------+
bool CyclesCreate(const long            chart_ID=0,         // ID графика
                  const string          name="Cycles",      // имя объекта
                  const int             sub_window=0,       // номер подокна
                  datetime              time1=0,            // время первой точки
                  double                price1=0,           // цена первой точки
                  datetime              time2=0,            // время второй точки
                  double                price2=0,           // цена второй точки
                  const color           clr=clrRed,         // цвет циклических линий
                  const ENUM_LINE_STYLE style=STYLE_SOLID,  // стиль циклических линий
                  const int             width=1,            // толщина циклических лин
                  const bool            back=false,         // на заднем плане
                  const bool            selection=true,     // выделить для перемещени
                  const bool            hidden=true,        // скрыт в списке объектов
                  const long            z_order=0)          // приоритет на нажатие мы
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeCyclesEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим циклические линии по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_CYCLES,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": не удалось создать циклические линии! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет линий
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линий
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линий мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool CyclesPointChange(const long   chart_ID=0,    // ID графика
                       const string name="Cycles", // имя объекта
                       const int    point_index=0, // номер точки привязки
                       datetime     time=0,        // координата времени точки привяз
                       double       price=0)       // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет циклические линии                                        |
//+------------------------------------------------------------------+
bool CyclesDelete(const long   chart_ID=0,    // ID графика
                  const string name="Cycles") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим циклические линии
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить циклические линии! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки циклических линий, и для пустых|
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeCyclesEmptyPoints(datetime &time1,double &price1,
                             datetime &time2,double &price2)
  {
//--- если время первой точки не задано, то она будет на текущем баре
   if(!time1)
      time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее второй
   if(!time2)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- установим вторую точку на 9 баров левее первой
      time2=temp[0];
     }
//--- если цена второй точки не задана, то она совпадает с ценой первой точки
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки циклических линий
   datetime date[];
   double   price[];
```

```
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования циклических линий
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- создадим линию тренда
   if(!CyclesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
   int h_steps=bars/5;
//--- перемещаем вторую точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d2<bars-1)
         d2+=1;
      //--- сдвигаем точку
      if(!CyclesPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
```

```
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   h_steps=bars/4;
//--- перемещаем первую точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d1<bars-1)
         d1+=1;
      //--- сдвигаем точку
      if(!CyclesPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим объект с графика
   CyclesDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROWED_LINE

Линия со стрелкой.

Пример

Следующий скрипт создает и перемещает на графике линию со стрелкой. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Линия со стрелкой\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="ArrowedLine"; // Имя линии
input int             InpDate1=35;           // Дата 1-ой точки в %
input int             InpPrice1=60;          // Цена 1-ой точки в %
input int             InpDate2=65;           // Дата 2-ой точки в %
input int             InpPrice2=40;          // Цена 2-ой точки в %
input color           InpColor=clrRed;       // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASH;   // Стиль линии
input int             InpWidth=2;            // Толщина линии
input bool            InpBack=false;         // Линия на заднем плане
input bool            InpSelection=true;     // Выделить для перемещений
input bool            InpHidden=true;        // Скрыт в списке объектов
```

```
input long            InpZOrder=0;           // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает линию со стрелкой по заданным координатам                |
//+------------------------------------------------------------------+
bool ArrowedLineCreate(const long            chart_ID=0,         // ID графика
                       const string          name="ArrowedLine", // имя линии
                       const int             sub_window=0,       // номер подокна
                       datetime              time1=0,            // время первой точки
                       double                price1=0,           // цена первой точки
                       datetime              time2=0,            // время второй точки
                       double                price2=0,           // цена второй точки
                       const color           clr=clrRed,         // цвет линии
                       const ENUM_LINE_STYLE style=STYLE_SOLID,  // стиль линии
                       const int             width=1,            // толщина линии
                       const bool            back=false,         // на заднем плане
                       const bool            selection=true,     // выделить для пере
                       const bool            hidden=true,        // скрыт в списке об
                       const long            z_order=0)          // приоритет на нажа
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeArrowedLineEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим линию со стрелкой по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_ARROWED_LINE,sub_window,time1,price1,time2,pric
     {
      Print(__FUNCTION__,
            ": не удалось создать линию со стрелкой! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет линии
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения линии мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
```

```
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки линии со стрелкой                      |
//+------------------------------------------------------------------+
bool ArrowedLinePointChange(const long   chart_ID=0,          // ID графика
                            const string name="ArrowedLine", // имя линии
                            const int    point_index=0,      // номер точки привязки
                            datetime     time=0,             // координата времени то
                            double       price=0)            // координата цены точки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки линии
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Функция удаляет линию со стрелкой с графика.                     |
//+------------------------------------------------------------------+
bool ArrowedLineDelete(const long   chart_ID=0,          // ID графика
                       const string name="ArrowedLine") // имя линии
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим линию со стрелкой
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить линию со стрелкой! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
```

```
//| Проверяет значения точек привязки линии, и для пустых значений   |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowedLineEmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2)
  {
//--- если время первой точки не задано, то она будет на текущем баре
   if(!time1)
      time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее первой
   if(!time2)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- установим вторую точку на 9 баров левее первой
      time2=temp[0];
     }
//--- если цена второй точки не задана, то она совпадает с ценой первой точки
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки линии
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
```

```
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования линии
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- создадим линию со стрелкой
   if(!ArrowedLineCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки линии
//--- счетчик цикла
   int v_steps=accuracy/5;
//--- перемещаем вторую точку привязки по вертикали
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p2<accuracy-1)
         p2+=1;
      //--- сдвигаем точку
      if(!ArrowedLinePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- перемещаем первую точку привязки по вертикали
```

```
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1>1)
         p1-=1;
      //--- сдвигаем точку
      if(!ArrowedLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в пол секунды
   Sleep(500);
//--- счетчик цикла
   int h_steps=bars/2;
//--- перемещаем обе точки привязки по горизонтали одновременно
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующие значения
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- сдвигаем точки
      if(!ArrowedLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!ArrowedLinePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.03 секунды
      Sleep(30);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим линию со стрелкой
   ArrowedLineDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_CHANNEL

Равноудаленный канал.

Примечание

Для равноудаленного канала можно указать режим продолжения его отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно). Также можно установить режим заливки канала цветом.

Пример

Следующий скрипт создает и перемещает на графике равноудаленный канал. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Равноудаленный канал\"."
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="Channel";   // Имя канала
input int             InpDate1=25;         // Дата 1-ой точки в %
input int             InpPrice1=60;        // Цена 1-ой точки в %
input int             InpDate2=65;         // Дата 2-ой точки в %
input int             InpPrice2=80;        // Цена 2-ой точки в %
input int             InpDate3=30;         // Дата 3-ей точки в %
```

```
input int              InpPrice3=40;         // Цена 3-ей точки в %
input color            InpColor=clrRed;      // Цвет канала
input ENUM_LINE_STYLE  InpStyle=STYLE_DASH;  // Стиль линий канала
input int              InpWidth=2;           // Толщина линий канала
input bool             InpBack=false;        // Канал на заднем плане
input bool             InpFill=false;        // Заливка канала цветом
input bool             InpSelection=true;    // Выделить для перемещений
input bool             InpRayLeft=false;     // Продолжение канала влево
input bool             InpRayRight=false;    // Продолжение канала вправо
input bool             InpHidden=true;       // Скрыт в списке объектов
input long             InpZOrder=0;          // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает равноудаленный канал по заданным координатам             |
//+------------------------------------------------------------------+
bool ChannelCreate(const long            chart_ID=0,        // ID графика
                   const string          name="Channel",    // имя канала
                   const int             sub_window=0,      // номер подокна
                   datetime              time1=0,           // время первой точки
                   double                price1=0,          // цена первой точки
                   datetime              time2=0,           // время второй точки
                   double                price2=0,          // цена второй точки
                   datetime              time3=0,           // время третьей точки
                   double                price3=0,          // цена третьей точки
                   const color           clr=clrRed,        // цвет канала
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий канала
                   const int             width=1,           // толщина линий канала
                   const bool            fill=false,        // заливка канала цветом
                   const bool            back=false,        // на заднем плане
                   const bool            selection=true,    // выделить для перемещени
                   const bool            ray_left=false,    // продолжение канала вле
                   const bool            ray_right=false,   // продолжение канала впр
                   const bool            hidden=true,       // скрыт в списке объекто
                   const long            z_order=0)         // приоритет на нажатие м
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим канал по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_CHANNEL,sub_window,time1,price1,time2,price2,ti
     {
      Print(__FUNCTION__,
            ": не удалось создать равноудаленный канал! Код ошибки = ",GetLastError()
      return(false);
     }
//--- установим цвет канала
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий канала
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий канала
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки канала
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения канала для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения канала влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения канала вправо
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки канала                                 |
//+------------------------------------------------------------------+
bool ChannelPointChange(const long   chart_ID=0,     // ID графика
                        const string name="Channel", // имя канала
                        const int    point_index=0,  // номер точки привязки
                        datetime     time=0,         // координата времени точки прив
                        double       price=0)        // координата цены точки привязк
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет канал                                                    |
//+------------------------------------------------------------------+
bool ChannelDelete(const long   chart_ID=0,     // ID графика
                   const string name="Channel") // имя канала
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим канал
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить канал! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки канала, и для пустых значений  |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                              double &price2,datetime &time3,double &price3)
  {
//--- если время второй (правой) точки не задано, то она будет на текущем баре
   if(!time2)
      time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой (левой) точки не задано, то она лежит на 9 баров левее второй
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то сдвинем ее на 300 пунктов выше второй
   if(!price1)
      price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время третьей точки не задано, то оно совпадает с временем первой точки
   if(!time3)
      time3=time1;
```

```
//--- если цена третьей точки не задана, то она совпадает с ценой второй точки
   if(!price3)
      price3=price2;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   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("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки канала
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования канала
   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;
//--- создадим равноудаленный канал
   if(!ChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
      InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки канала
//--- счетчик цикла
   int h_steps=bars/6;
//--- перемещаем вторую точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d2<bars-1)
         d2+=1;
      //--- сдвигаем точку
      if(!ChannelPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- перемещаем первую точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d1>1)
         d1-=1;
      //--- сдвигаем точку
      if(!ChannelPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
```

```
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int v_steps=accuracy/10;
//--- перемещаем третью точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p3>1)
         p3-=1;
      //--- сдвигаем точку
      if(!ChannelPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим канал с графика
   ChannelDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_STDDEVCHANNEL

Канал стандартного отклонения.

Примечание

Для канала стандартного отклонения можно указать режим продолжения его отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно). Также можно установить режим заливки канала цветом.

Для изменения значения отклонения канала используется свойство OBJPROP_DEVIATION.

Пример

Следующий скрипт создает и перемещает на графике канал стандартного отклонения. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Канал стандартного отклонени
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="StdDevChannel";   // Имя канала
input int              InpDate1=10;               // Дата 1-ой точки в %
input int              InpDate2=40;               // Дата 2-ой точки в %
input double           InpDeviation=1.0;          // Отклонение
input color            InpColor=clrRed;           // Цвет канала
```

```
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий канала
input int             InpWidth=2;                // Толщина линий канала
input bool            InpFill=false;             // Заливка канала цветом
input bool            InpBack=false;             // Канал на заднем плане
input bool            InpSelection=true;         // Выделить для перемещений
input bool            InpRayLeft=false;          // Продолжение канала влево
input bool            InpRayRight=false;         // Продолжение канала вправо
input bool            InpHidden=true;            // Скрыт в списке объектов
input long            InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает канал стандартного отклонения по заданным координатам    |
//+------------------------------------------------------------------+
bool StdDevChannelCreate(const long            chart_ID=0,        // ID графика
                         const string          name="Channel",    // имя канала
                         const int             sub_window=0,      // номер подокна
                         datetime              time1=0,           // время первой точ
                         datetime              time2=0,           // время второй точ
                         const double          deviation=1.0,     // отклонение
                         const color           clr=clrRed,        // цвет канала
                         const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий кана
                         const int             width=1,           // толщина линий ка
                         const bool            fill=false,        // заливка канала ц
                         const bool            back=false,        // на заднем плане
                         const bool            selection=true,    // выделить для пер
                         const bool            ray_left=false,    // продолжение кана
                         const bool            ray_right=false,   // продолжение кана
                         const bool            hidden=true,       // скрыт в списке о
                         const long            z_order=0)         // приоритет на наж
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeChannelEmptyPoints(time1,time2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим канал по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_STDDEVCHANNEL,sub_window,time1,0,time2,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать канал стандартного отклонения! Код ошибки = ",GetLa
      return(false);
     }
//--- установим величину отклонения, от нее зависит ширина канала
   ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation);
//--- установим цвет канала
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий канала
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий канала
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
```

```
//--- включим (true) или отключим (false) режим заливки канала
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения канала для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения канала влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения канала вправо
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки канала                                 |
//+------------------------------------------------------------------+
bool StdDevChannelPointChange(const long   chart_ID=0,     // ID графика
                              const string name="Channel", // имя канала
                              const int    point_index=0,  // номер точки привязки
                              datetime     time=0)         // координата времени точки
  {
//--- если время точки не задано, то перемещаем ее на текущий бар
   if(!time)
      time=TimeCurrent();
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,0))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет отклонение канала                                       |
//+------------------------------------------------------------------+
bool StdDevChannelDeviationChange(const long   chart_ID=0,     // ID графика
```

```
                                    const string name="Channel", // имя канала
                                    const double deviation=1.0)  // отклонение
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим угол наклона линии тренда
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_DEVIATION,deviation))
     {
      Print(__FUNCTION__,
            ": не удалось изменить отклонение канала! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет канал                                                    |
//+------------------------------------------------------------------+
bool StdDevChannelDelete(const long   chart_ID=0,     // ID графика
                         const string name="Channel") // имя канала
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим канал
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить канал! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки канала, и для пустых значений  |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,datetime &time2)
  {
//--- если время второй точки не задано, то она будет на текущем баре
   if(!time2)
      time2=TimeCurrent();
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
```

```
      //--- установим вторую точку на 9 баров левее второй
      time1=temp[0];
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки канала
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования канала
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
//--- создадим канал стандартного отклонения
   if(!StdDevChannelCreate(0,InpName,0,date[d1],date[d2],InpDeviation,InpColor,InpSty
      InpWidth,InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrde
```

```
      {
       return;
      }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать канал по горизонтали вправо и расширять его
//--- счетчик цикла
   int h_steps=bars/2;
//--- перемещаем канал
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующие значения
      if(d1<bars-1)
         d1+=1;
      if(d2<bars-1)
         d2+=1;
      //--- переместим точки привязки
      if(!StdDevChannelPointChange(0,InpName,0,date[d1]))
         return;
      if(!StdDevChannelPointChange(0,InpName,1,date[d2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   double v_steps=InpDeviation*2;
//--- расширим канал
   for(double i=InpDeviation;i<v_steps;i+=10.0/accuracy)
     {
      if(!StdDevChannelDeviationChange(0,InpName,i))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим канал с графика
```

```
   StdDevChannelDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_REGRESSION

Канал на линейной регрессии.

Примечание

Для канала на линейной регрессии можно указать режим продолжения его отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно). Также можно установить режим заливки канала цветом.

Пример

Следующий скрипт создает и перемещает на графике канал на линейной регрессии. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Канал на линейной регрессии
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="Regression"; // Имя канала
input int             InpDate1=10;          // Дата 1-ой точки в %
input int             InpDate2=40;          // Дата 2-ой точки в %
input color           InpColor=clrRed;      // Цвет канала
input ENUM_LINE_STYLE InpStyle=STYLE_DASH;  // Стиль линий канала
input int             InpWidth=2;           // Толщина линий канала
```

```
input bool              InpFill=false;         // Заливка канала цветом
input bool              InpBack=false;         // Канал на заднем плане
input bool              InpSelection=true;     // Выделить для перемещений
input bool              InpRayLeft=false;      // Продолжение канала влево
input bool              InpRayRight=false;     // Продолжение канала вправо
input bool              InpHidden=true;        // Скрыт в списке объектов
input long              InpZOrder=0;           // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает канал на линейной регрессии по заданным координатам      |
//+------------------------------------------------------------------+
bool RegressionCreate(const long            chart_ID=0,        // ID графика
                      const string          name="Regression", // имя канала
                      const int             sub_window=0,      // номер подокна
                      datetime              time1=0,           // время первой точки
                      datetime              time2=0,           // время второй точки
                      const color           clr=clrRed,        // цвет канала
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий канала
                      const int             width=1,           // толщина линий канал
                      const bool            fill=false,        // заливка канала цвет
                      const bool            back=false,        // на заднем плане
                      const bool            selection=true,    // выделить для переме
                      const bool            ray_left=false,    // продолжение канала
                      const bool            ray_right=false,   // продолжение канала
                      const bool            hidden=true,       // скрыт в списке объе
                      const long            z_order=0)         // приоритет на нажати
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeRegressionEmptyPoints(time1,time2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим канал по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_REGRESSION,sub_window,time1,0,time2,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать канал на линейной регрессии! Код ошибки = ",GetLast
      return(false);
     }
//--- установим цвет канала
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий канала
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий канала
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки канала
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения канала для перемещений
```

```
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения канала влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения канала вправо
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки канала                                 |
//+------------------------------------------------------------------+
bool RegressionPointChange(const long   chart_ID=0,      // ID графика
                           const string name="Channel", // имя канала
                           const int    point_index=0,  // номер точки привязки
                           datetime     time=0)         // координата времени точки п
  {
//--- если время точки не задано, то перемещаем ее на текущий бар
   if(!time)
      time=TimeCurrent();
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,0))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет канал                                                    |
//+------------------------------------------------------------------+
bool RegressionDelete(const long   chart_ID=0,      // ID графика
                      const string name="Channel") // имя канала
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим канал
```

```
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить канал! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки канала, и для пустых значений  |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeRegressionEmptyPoints(datetime &time1,datetime &time2)
  {
//--- если время второй точки не задано, то она будет на текущем баре
   if(!time2)
      time2=TimeCurrent();
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки канала
   datetime date[];
   double   price[];
```

```
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования канала
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
//--- создадим канал на линейной регрессии
   if(!RegressionCreate(0,InpName,0,date[d1],date[d2],InpColor,InpStyle,InpWidth,
      InpFill,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать канал по горизонтали вправо
//--- счетчик цикла
   int h_steps=bars/2;
//--- перемещаем канал
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующие значения
      if(d1<bars-1)
         d1+=1;
      if(d2<bars-1)
         d2+=1;
      //--- переместим точки привязки
      if(!RegressionPointChange(0,InpName,0,date[d1]))
         return;
      if(!RegressionPointChange(0,InpName,1,date[d2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
```

```
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим канал с графика
   RegressionDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_PITCHFORK

Вилы Эндрюса.

Примечание

Для "Вил Эндрюса" можно указать режим продолжения отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно).

Также можно указать количество линий-уровня, их значения и цвет.

Пример

Следующий скрипт создает и перемещает на графике "Вилы Эндрюса". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Вилы Эндрюса\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="Pitchfork";  // Имя вил
input int             InpDate1=14;          // Дата 1-ой точки в %
input int             InpPrice1=40;         // Цена 1-ой точки в %
input int             InpDate2=18;          // Дата 2-ой точки в %
input int             InpPrice2=50;         // Цена 2-ой точки в %
input int             InpDate3=18;          // Дата 3-ей точки в %
```

```
input int              InpPrice3=30;          // Цена 3-ей точки в %
input color            InpColor=clrRed;       // Цвет вил
input ENUM_LINE_STYLE  InpStyle=STYLE_SOLID;  // Стиль линий вил
input int              InpWidth=1;            // Толщина линий вил
input bool             InpBack=false;         // Вилы на заднем плане
input bool             InpSelection=true;     // Выделить для перемещений
input bool             InpRayLeft=false;      // Продолжение вил влево
input bool             InpRayRight=false;     // Продолжение вил вправо
input bool             InpHidden=true;        // Скрыт в списке объектов
input long             InpZOrder=0;           // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Вилы Эндрюса" по заданным координатам                   |
//+------------------------------------------------------------------+
bool PitchforkCreate(const long            chart_ID=0,        // ID графика
                     const string          name="Pitchfork",  // имя вил
                     const int             sub_window=0,      // номер подокна
                     datetime              time1=0,           // время первой точки
                     double                price1=0,          // цена первой точки
                     datetime              time2=0,           // время второй точки
                     double                price2=0,          // цена второй точки
                     datetime              time3=0,           // время третьей точки
                     double                price3=0,          // цена третьей точки
                     const color           clr=clrRed,        // цвет линий вил
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий вил
                     const int             width=1,           // толщина линий вил
                     const bool            back=false,        // на заднем плане
                     const bool            selection=true,    // выделить для перемещ
                     const bool            ray_left=false,    // продолжение вил влев
                     const bool            ray_right=false,   // продолжение вил впра
                     const bool            hidden=true,       // скрыт в списке объек
                     const long            z_order=0)         // приоритет на нажатие
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Вилы Эндрюса" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_PITCHFORK,sub_window,time1,price1,time2,price2,
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Вилы Эндрюса\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения вил для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения вил влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения вил вправо
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Задает количество уровней "Вил Эндрюса" и их параметры           |
//+------------------------------------------------------------------+
bool PitchforkLevelsSet(int             levels,           // количество линий уровня
                        double          &values[],        // значения линий уровня
                        color           &colors[],        // цвет линий уровня
                        ENUM_LINE_STYLE &styles[],        // стиль линий уровня
                        int             &widths[],        // толщина линий уровня
                        const long      chart_ID=0,       // ID графика
                        const string    name="Pitchfork") // имя вил
  {
//--- проверим размеры массивов
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": длина массива не соответствуют количеству уровней, ошибка
      return(false);
     }
//--- установим количество уровней
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
   for(int i=0;i<levels;i++)
     {
      //--- значение уровня
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- цвет уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- стиль уровня
```

```
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- толщина уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- описание уровня
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Вил Эндрюса"                          |
//+------------------------------------------------------------------+
bool PitchforkPointChange(const long   chart_ID=0,       // ID графика
                          const string name="Pitchfork", // имя канала
                          const int    point_index=0,    // номер точки привязки
                          datetime     time=0,           // координата времени точки
                          double       price=0)          // координата цены точки при
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет "Вилы Эндрюса"                                           |
//+------------------------------------------------------------------+
bool PitchforkDelete(const long   chart_ID=0,       // ID графика
                     const string name="Pitchfork") // имя канала
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим канал
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Вилы Эндрюса\"! Код ошибки = ",GetLastError());
```

```
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Вил Эндрюса", и для пустых    |
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                              double &price2,datetime &time3,double &price3)
  {
//--- если время второй (правой верхней) точки не задано, то она будет на текущем бар
   if(!time2)
      time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой (левой) точки не задано, то она лежит на 9 баров левее второй
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то сдвинем ее на 200 пунктов ниже второй
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время третьей точки не задано, то оно совпадает с временем второй точки
   if(!time3)
      time3=time2;
//--- если цена третьей точки не задана, то сдвинем ее на 200 пунктов ниже первой
   if(!price3)
      price3=price1-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   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("Ошибка! Некорректные значения входных параметров!");
```

```
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки "Вил Эндрюса"
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования "Вил Эндрюса"
   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;
//--- создадим вилы
   if(!PitchforkCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],pri
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidde
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки вил
//--- счетчик цикла
   int v_steps=accuracy/10;
//--- перемещаем первую точку привязки
```

```
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1>1)
         p1-=1;
      //--- сдвигаем точку
      if(!PitchforkPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int h_steps=bars/8;
//--- перемещаем третью точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d3<bars-1)
         d3+=1;
      //--- сдвигаем точку
      if(!PitchforkPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   v_steps=accuracy/10;
//--- перемещаем вторую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p2>1)
         p2-=1;
      //--- сдвигаем точку
```

```
      if(!PitchforkPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим вилы с графика
   PitchforkDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_GANNLINE

Линия Ганна.

Примечание

Для "Линии Ганна" можно указать режим продолжения ее отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно).

Для установки наклона линии можно использовать как угол Ганна с масштабом, так и координаты второй точки привязки.

Пример

Следующий скрипт создает и перемещает на графике "Линию Ганна". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Линия Ганна\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="GannLine";         // Имя линии
input int              InpDate1=20;                // Дата 1-ой точки в %
input int              InpPrice1=75;               // Цена 1-ой точки в %
input int              InpDate2=80;                // Дата 2-ой точки в %
input double           InpAngle=0.0;               // Угол Ганна
```

```
input double          InpScale=1.0;              // Масштаб
input color           InpColor=clrRed;           // Цвет линии
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int             InpWidth=2;                // Толщина линии
input bool            InpBack=false;             // Линия на заднем плане
input bool            InpSelection=true;         // Выделить для перемещений
input bool            InpRayLeft=false;          // Продолжение линии влево
input bool            InpRayRight=true;          // Продолжение линии вправо
input bool            InpHidden=true;            // Скрыт в списке объектов
input long            InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Линию Ганна" по координатам, углу и масштабу            |
//+------------------------------------------------------------------+
bool GannLineCreate(const long            chart_ID=0,        // ID графика
                    const string          name="GannLine",   // имя линии
                    const int             sub_window=0,      // номер подокна
                    datetime              time1=0,           // время первой точки
                    double                price1=0,          // цена первой точки
                    datetime              time2=0,           // время второй точки
                    const double          angle=1.0,         // угол Ганна
                    const double          scale=1.0,         // масштаб
                    const color           clr=clrRed,        // цвет линии
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии
                    const int             width=1,           // толщина линии
                    const bool            back=false,        // на заднем плане
                    const bool            selection=true,    // выделить для перемеще
                    const bool            ray_left=false,    // продолжение линии вле
                    const bool            ray_right=true,    // продолжение линии впр
                    const bool            hidden=true,       // скрыт в списке объект
                    const long            z_order=0)         // приоритет на нажатие
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeGannLineEmptyPoints(time1,price1,time2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Линию Ганна" по заданным координатам
//--- правильная координата цены второй точки привязки переопределится
//--- автоматически после изменения угла Ганна и (или) масштаба,
   if(!ObjectCreate(chart_ID,name,OBJ_GANNLINE,sub_window,time1,price1,time2,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Линию Ганна\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- изменим угол Ганна
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- изменим масштаб (количество пипсов на бар)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
```

```
//--- установим цвет линии
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения линии для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения линии влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения линии вправо
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Линии Ганна"                          |
//+------------------------------------------------------------------+
bool GannLinePointChange(const long   chart_ID=0,       // ID графика
                         const string name="GannLine", // имя линии
                         const int    point_index=0,   // номер точки привязки
                         datetime     time=0,          // координата времени точки пр
                         double       price=0)         // координата цены точки привя
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки линии
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
```

```
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет угол Ганна                                              |
//+------------------------------------------------------------------+
bool GannLineAngleChange(const long   chart_ID=0,      // ID графика
                         const string name="GannLine", // имя линии
                         const double angle=1.0)       // угол Ганна
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим угол Ганна
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle))
     {
      Print(__FUNCTION__,
            ": не удалось изменить угол Ганна! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет масштаб "Линии Ганна"                                   |
//+------------------------------------------------------------------+
bool GannLineScaleChange(const long   chart_ID=0,      // ID графика
                         const string name="GannLine", // имя линии
                         const double scale=1.0)       // масштаб
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим масштаб (количество пипсов на бар)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
      Print(__FUNCTION__,
            ": не удалось изменить масштаб! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Функция удаляет "Линию Ганна" с графика                          |
//+------------------------------------------------------------------+
bool GannLineDelete(const long   chart_ID=0,      // ID графика
                    const string name="GannLine") // имя линии
  {
//--- сбросим значение ошибки
```

```
   ResetLastError();
//--- удалим линию Ганна
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Линию Ганна\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Линии Ганна", и для пустых    |
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeGannLineEmptyPoints(datetime &time1,double &price1,datetime &time2)
  {
//--- если время второй точки не задано, то она будет на текущем баре
   if(!time2)
      time2=TimeCurrent();
//--- если время первой точки не задано, то она лежит на 9 баров левее первой
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
```

```
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки линии
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования линии Ганна
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- создадим линию Ганна
   if(!GannLineCreate(0,InpName,0,date[d1],price[p1],date[d2],InpAngle,InpScale,InpCo
      InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrd
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки линии и менять угол
//--- счетчик цикла
   int v_steps=accuracy/2;
//--- перемещаем первую точку привязки по вертикали
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1>1)
         p1-=1;
      //--- сдвигаем точку
      if(!GannLinePointChange(0,InpName,0,date[d1],price[p1]))
         return;
```

```
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в пол секунды
   Sleep(500);
//--- определим текущее значение угла Ганна (изменился,
//--- после перемещения первой точки привязки)
   double curr_angle;
   if(!ObjectGetDouble(0,InpName,OBJPROP_ANGLE,0,curr_angle))
      return;
//--- счетчик цикла
   v_steps=accuracy/8;
//--- изменяем угол Ганна
   for(int i=0;i<v_steps;i++)
     {
      if(!GannLineAngleChange(0,InpName,curr_angle-0.05*i))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим линию с графика
   GannLineDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_GANNFAN

Веер Ганна.

Примечание

Для "Веера Ганна" можно указать тип тренда из перечисления ENUM_GANN_DIRECTION. Регулируя значение масштаба (OBJPROP_SCALE), можно менять угол наклона линий веера.

Пример

Следующий скрипт создает и перемещает на графике "Веер Ганна". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Веер Ганна\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="GannFan";          // Имя веера
input int             InpDate1=15;                // Дата 1-ой точки в %
input int             InpPrice1=25;               // Цена 1-ой точки в %
input int             InpDate2=85;                // Дата 2-ой точки в %
input double          InpScale=2.0;               // Масштаб
input bool            InpDirection=false;         // Направление тренда
input color           InpColor=clrRed;            // Цвет веера
```

```
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий веера
input int             InpWidth=1;                // Толщина линий веера
input bool            InpBack=false;             // Веер на заднем плане
input bool            InpSelection=true;         // Выделить для перемещений
input bool            InpHidden=true;            // Скрыт в списке объектов
input long            InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Веер Ганна"                                             |
//+------------------------------------------------------------------+
bool GannFanCreate(const long            chart_ID=0,        // ID графика
                   const string          name="GannFan",    // имя веера
                   const int             sub_window=0,      // номер подокна
                   datetime              time1=0,           // время первой точки
                   double                price1=0,          // цена первой точки
                   datetime              time2=0,           // время второй точки
                   const double          scale=1.0,         // масштаб
                   const bool            direction=true,    // направление тренда
                   const color           clr=clrRed,        // цвет веера
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий веера
                   const int             width=1,           // толщина линий веера
                   const bool            back=false,        // на заднем плане
                   const bool            selection=true,    // выделить для перемещен
                   const bool            hidden=true,       // скрыт в списке объекто
                   const long            z_order=0)         // приоритет на нажатие м
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeGannFanEmptyPoints(time1,price1,time2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Веер Ганна" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_GANNFAN,sub_window,time1,price1,time2,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Веер Ганна\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- изменим масштаб (количество пипсов на бар)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- изменим направление тренда "Веера Ганна" (true - нисходящий, false - восходящий
   ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
//--- установим цвет веера
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линий веера
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий веера
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

```
//--- включим (true) или отключим (false) режим выделения веера для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Веера Ганна"                          |
//+------------------------------------------------------------------+
bool GannFanPointChange(const long   chart_ID=0,     // ID графика
                        const string name="GannFan", // имя веера
                        const int    point_index=0,  // номер точки привязки
                        datetime     time=0,         // координата времени точки прив
                        double       price=0)        // координата цены точки привязк
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки веера
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет масштаб "Веера Ганна"                                   |
//+------------------------------------------------------------------+
bool GannFanScaleChange(const long   chart_ID=0,     // ID графика
                        const string name="GannFan", // имя веера
                        const double scale=1.0)      // масштаб
  {
//--- сбросим значение ошибки
   ResetLastError();
```

```
//--- изменим масштаб (количество пипсов на бар)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
      Print(__FUNCTION__,
            ": не удалось изменить масштаб! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет направление тренда "Веера Ганна"                        |
//+------------------------------------------------------------------+
bool GannFanDirectionChange(const long   chart_ID=0,     // ID графика
                            const string name="GannFan", // имя веера
                            const bool   direction=true) // направление тренда
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим направление тренда "Веера Ганна"
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
     {
      Print(__FUNCTION__,
            ": не удалось изменить направление тренда! Код ошибки = ",GetLastError())
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Функция удаляет "Веер Ганна" с графика                           |
//+------------------------------------------------------------------+
bool GannFanDelete(const long   chart_ID=0,     // ID графика
                   const string name="GannFan") // имя веера
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим веер Ганна
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Веер Ганна\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
```

```
//| Проверяет значения точек привязки "Веера Ганна", и для пустых    |
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeGannFanEmptyPoints(datetime &time1,double &price1,datetime &time2)
  {
//--- если время второй точки не задано, то она будет на текущем баре
   if(!time2)
      time2=TimeCurrent();
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки веера
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
```

```
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования веера Ганна
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- создадим веер Ганна
   if(!GannFanCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки веера
//--- счетчик цикла
   int v_steps=accuracy/2;
//--- перемещаем первую точку привязки по вертикали
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1<accuracy-1)
         p1+=1;
      //--- сдвигаем точку
      if(!GannFanPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- изменим направление тренда веера на нисходящее
   GannFanDirectionChange(0,InpName,true);
//--- перерисуем график
   ChartRedraw();
```

```
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим веер с графика
   GannFanDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_GANNGRID

Сетка Ганна.

Примечание

Для "Сетки Ганна" можно указать тип тренда из перечисления ENUM_GANN_DIRECTION. Регулируя значение масштаба (OBJPROP_SCALE), можно менять угол наклона линий сетки.

Пример

Следующий скрипт создает и перемещает на графике "Сетку Ганна". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Сетка Ганна\"."
#property description "Координаты точек привязки сетки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="GannGrid";        // Имя сетки
input int             InpDate1=15;               // Дата 1-ой точки в %
input int             InpPrice1=25;              // Цена 1-ой точки в %
input int             InpDate2=35;               // Дата 2-ой точки в %
input double          InpScale=3.0;              // Масштаб
input bool            InpDirection=false;        // Направление тренда
input color           InpColor=clrRed;           // Цвет сетки
```

```
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий сетки
input int             InpWidth=1;                // Толщина линий веера
input bool            InpBack=false;             // Сетка на заднем плане
input bool            InpSelection=true;         // Выделить для перемещений
input bool            InpHidden=true;            // Скрыт в списке объектов
input long            InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Сетку Ганна"                                            |
//+------------------------------------------------------------------+
bool GannGridCreate(const long            chart_ID=0,        // ID графика
                    const string          name="GannGrid",   // имя сетки
                    const int             sub_window=0,      // номер подокна
                    datetime              time1=0,           // время первой точки
                    double                price1=0,          // цена первой точки
                    datetime              time2=0,           // время второй точки
                    const double          scale=1.0,         // масштаб
                    const bool            direction=true,    // направление тренда
                    const color           clr=clrRed,        // цвет сетки
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий сетки
                    const int             width=1,           // толщина линий сетки
                    const bool            back=false,        // на заднем плане
                    const bool            selection=true,    // выделить для перемеще
                    const bool            hidden=true,       // скрыт в списке объект
                    const long            z_order=0)         // приоритет на нажатие
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeGannGridEmptyPoints(time1,price1,time2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Сетка Ганна" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_GANNGRID,sub_window,time1,price1,time2,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Сетку Ганна\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- изменим масштаб (количество пипсов на бар)
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- изменим направление тренда "Сетки Ганна" (true - нисходящий, false - восходящий
   ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction);
//--- установим цвет сетки
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль отображения линий сетки
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий сетки
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

```
//--- включим (true) или отключим (false) режим выделения сетки для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Сетки Ганна"                          |
//+------------------------------------------------------------------+
bool GannGridPointChange(const long   chart_ID=0,      // ID графика
                         const string name="GannGrid", // имя сетки
                         const int    point_index=0,   // номер точки привязки
                         datetime     time=0,          // координата времени точки пр
                         double       price=0)         // координата цены точки привя
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки ctnrb
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет масштаб "Сетки Ганна"                                   |
//+------------------------------------------------------------------+
bool GannGridScaleChange(const long   chart_ID=0,      // ID графика
                         const string name="GannGrid", // сетки
                         const double scale=1.0)       // масштаб
  {
//--- сбросим значение ошибки
   ResetLastError();
```

```
//--- изменим масштаб (количество пипсов на бар)
   if(!ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale))
     {
      Print(__FUNCTION__,
            ": не удалось изменить масштаб! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет направление тренда "Сетки Ганна"                        |
//+------------------------------------------------------------------+
bool GannGridDirectionChange(const long   chart_ID=0,      // ID графика
                             const string name="GannGrid", // имя сетки
                             const bool   direction=true)  // направление тренда
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим направление тренда "Сетки Ганна"
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_DIRECTION,direction))
     {
      Print(__FUNCTION__,
            ": не удалось изменить направление тренда! Код ошибки = ",GetLastError())
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Функция удаляет "Сетку Ганна" с графика                          |
//+------------------------------------------------------------------+
bool GannGridDelete(const long   chart_ID=0,      // ID графика
                    const string name="GannGrid") // имя сетки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим сетку Ганна
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Сетку Ганна\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
```

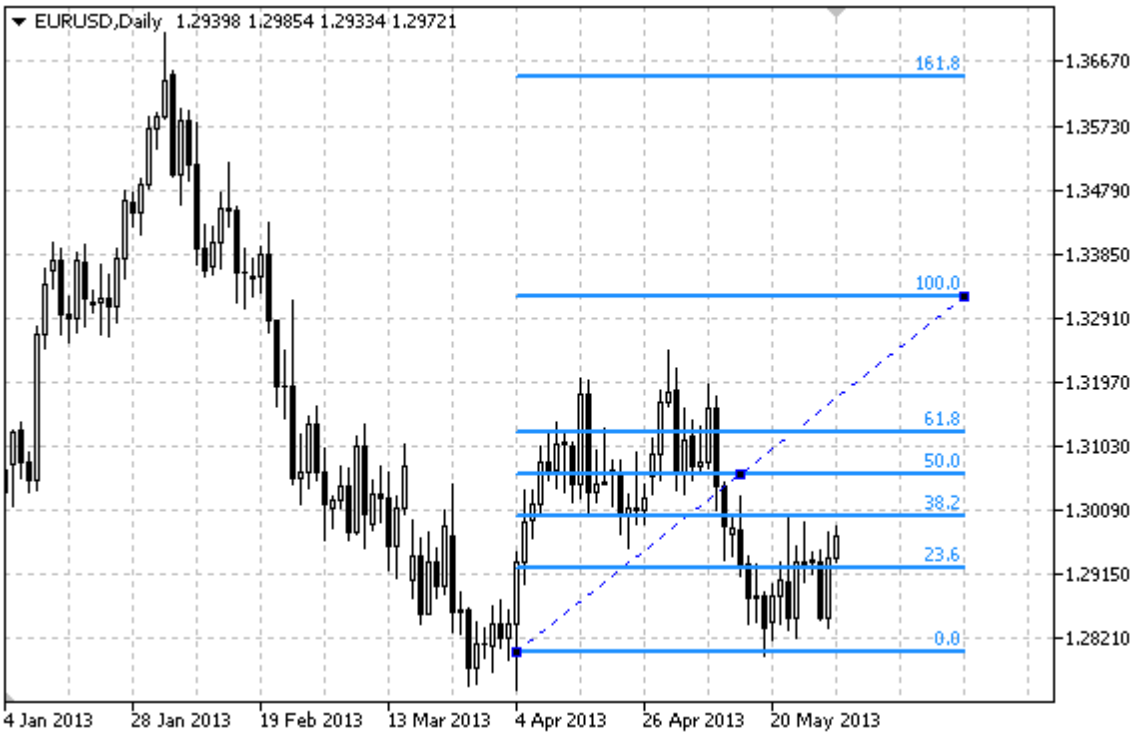
```
//| Проверяет значения точек привязки "Сетки Ганна", и для пустых    |
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeGannGridEmptyPoints(datetime &time1,double &price1,datetime &time2)
  {
//--- если время второй точки не задано, то она будет на текущем баре
   if(!time2)
      time2=TimeCurrent();
//--- если время первой точки не задано, то она лежит на 9 баров левее первой
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки сетки
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
```

```
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования сетки Ганна
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
//--- создадим сетку Ганна
   if(!GannGridCreate(0,InpName,0,date[d1],price[p1],date[d2],InpScale,InpDirection,
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки сетки
//--- счетчик цикла
   int v_steps=accuracy/4;
//--- перемещаем первую точку привязки по вертикали
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1<accuracy-1)
         p1+=1;
      if(!GannGridPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int h_steps=bars/4;
//--- перемещаем вторую точку привязки по горизонтали
   for(int i=0;i<h_steps;i++)
     {
```

```
      //--- возьмем следующее значение
      if(d2<bars-1)
         d2+=1;
      if(!GannGridPointChange(0,InpName,1,date[d2],0))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- изменим направление тренда сетки на нисходящее
   GannGridDirectionChange(0,InpName,true);
//--- перерисуем график
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим сетку с графика
   GannGridDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_FIBO

Уровни Фибоначчи.



Примечание

Для "Уровней Фибоначчи" можно указать режим продолжения отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно).

Также можно указать количество линий-уровня, их значения и цвет.

Пример

Следующий скрипт создает и перемещает на графике "Уровни Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Уровни Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="FiboLevels";      // Имя объекта
input int             InpDate1=10;               // Дата 1-ой точки в %
input int             InpPrice1=65;              // Цена 1-ой точки в %
input int             InpDate2=90;               // Дата 2-ой точки в %
input int             InpPrice2=85;              // Цена 2-ой точки в %
input color           InpColor=clrRed;           // Цвет объекта
```

```
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int             InpWidth=2;                // Толщина линии
input bool            InpBack=false;             // Объект на заднем плане
input bool            InpSelection=true;         // Выделить для перемещений
input bool            InpRayLeft=false;          // Продолжение объекта влево
input bool            InpRayRight=false;         // Продолжение объекта вправо
input bool            InpHidden=true;            // Скрыт в списке объектов
input long            InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Уровни Фибоначчи" по заданным координатам               |
//+------------------------------------------------------------------+
bool FiboLevelsCreate(const long            chart_ID=0,        // ID графика
                      const string          name="FiboLevels", // имя объекта
                      const int             sub_window=0,      // номер подокна
                      datetime              time1=0,           // время первой точки
                      double                price1=0,          // цена первой точки
                      datetime              time2=0,           // время второй точки
                      double                price2=0,          // цена второй точки
                      const color           clr=clrRed,        // цвет объекта
                      const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии объекта
                      const int             width=1,           // толщина линии объек
                      const bool            back=false,        // на заднем плане
                      const bool            selection=true,    // выделить для переме
                      const bool            ray_left=false,    // продолжение объекта
                      const bool            ray_right=false,   // продолжение объекта
                      const bool            hidden=true,       // скрыт в списке объе
                      const long            z_order=0)         // приоритет на нажати
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeFiboLevelsEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Уровни Фибоначчи" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_FIBO,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Уровни Фибоначчи\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения объекта для перемещений
```

```
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения объекта влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения объекта вправ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Задает количество уровней и их параметры                         |
//+------------------------------------------------------------------+
bool FiboLevelsSet(int              levels,           // количество линий уровня
                   double           &values[],        // значения линий уровня
                   color            &colors[],        // цвет линий уровня
                   ENUM_LINE_STYLE &styles[],         // стиль линий уровня
                   int              &widths[],        // толщина линий уровня
                   const long       chart_ID=0,       // ID графика
                   const string     name="FiboLevels") // имя объекта
  {
//--- проверим размеры массивов
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": длина массива не соответствуют количеству уровней, ошибка
      return(false);
     }
//--- установим количество уровней
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
   for(int i=0;i<levels;i++)
     {
      //--- значение уровня
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- цвет уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- стиль уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- толщина уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- описание уровня
```

```
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Уровней Фибоначчи"                    |
//+------------------------------------------------------------------+
bool FiboLevelsPointChange(const long   chart_ID=0,        // ID графика
                           const string name="FiboLevels", // имя объекта
                           const int    point_index=0,     // номер точки привязки
                           datetime     time=0,            // координата времени точки
                           double       price=0)           // координата цены точки п
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет "Уровни Фибоначчи"                                       |
//+------------------------------------------------------------------+
bool FiboLevelsDelete(const long   chart_ID=0,        // ID графика
                      const string name="FiboLevels") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим объект
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Уровни Фибоначчи\"! Код ошибки = ",GetLastError()
      return(false);
     }
//--- успешное выполнение
   return(true);
```

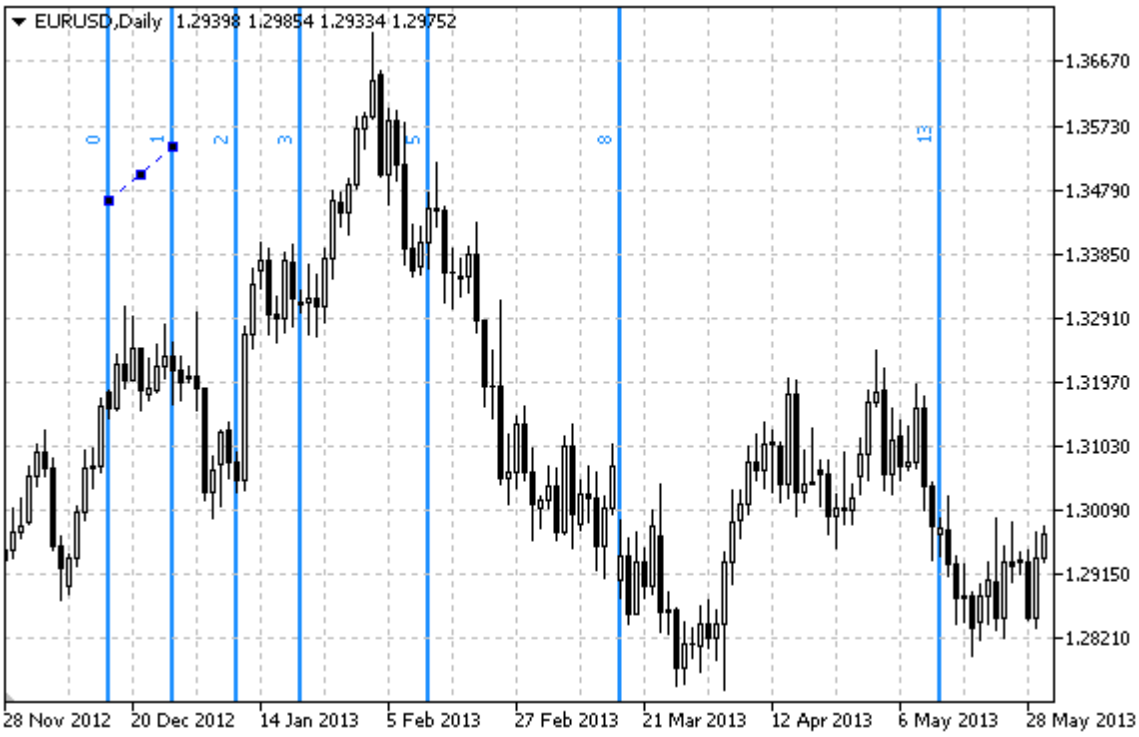
```
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Уровней Фибоначчи", и для     |
//| пустых значений устанавливает значения по умолчанию              |
//+------------------------------------------------------------------+
void ChangeFiboLevelsEmptyPoints(datetime &time1,double &price1,
                                 datetime &time2,double &price2)
  {
//--- если время второй точки не задано, то она будет на текущем баре
   if(!time2)
      time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то сдвинем ее на 200 пунктов ниже второй
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки "Уровней Фибоначчи"
   datetime date[];
   double   price[];
//--- выделение памяти
```

```
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования "Уровней Фибоначчи"
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- создадим объект
   if(!FiboLevelsCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidden,InpZOrd
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
   int v_steps=accuracy*2/5;
//--- перемещаем первую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1>1)
         p1-=1;
      //--- сдвигаем точку
      if(!FiboLevelsPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
```

```
      }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   v_steps=accuracy*4/5;
//--- перемещаем вторую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p2>1)
         p2-=1;
      //--- сдвигаем точку
      if(!FiboLevelsPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим объект с графика
   FiboLevelsDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_FIBOTIMES

Временные зоны Фибоначчи.



Примечание

Для "Временных зон Фибоначчи" можно указать количество линий-уровня, их значения и цвет.

Пример

Следующий скрипт создает и перемещает на графике "Временные зоны Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Временные зоны Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="FiboTimes";       // Имя объекта
input int              InpDate1=10;               // Дата 1-ой точки в %
input int              InpPrice1=45;              // Цена 1-ой точки в %
input int              InpDate2=20;               // Дата 2-ой точки в %
input int              InpPrice2=55;              // Цена 2-ой точки в %
input color            InpColor=clrRed;           // Цвет объекта
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int              InpWidth=2;                // Толщина линии
input bool             InpBack=false;             // Объект на заднем плане
```

```
input bool            InpSelection=true;         // Выделить для перемещений
input bool            InpHidden=true;            // Скрыт в списке объектов
input long            InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Временные зоны Фибоначчи" по заданным координатам       |
//+------------------------------------------------------------------+
bool FiboTimesCreate(const long            chart_ID=0,        // ID графика
                     const string          name="FiboTimes",  // имя объекта
                     const int             sub_window=0,      // номер подокна
                     datetime              time1=0,           // время первой точки
                     double                price1=0,          // цена первой точки
                     datetime              time2=0,           // время второй точки
                     double                price2=0,          // цена второй точки
                     const color           clr=clrRed,        // цвет объекта
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии объекта
                     const int             width=1,           // толщина линии объект
                     const bool            back=false,        // на заднем плане
                     const bool            selection=true,    // выделить для перемещ
                     const bool            hidden=true,       // скрыт в списке объек
                     const long            z_order=0)         // приоритет на нажатие
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeFiboTimesEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Временные зоны Фибоначчи" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOTIMES,sub_window,time1,price1,time2,price2)
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Временные зоны Фибоначчи\"! Код ошибки = ",GetLas
      return(false);
     }
//--- установим цвет
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения объекта для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
```

```
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Задает количество уровней и их параметры                         |
//+------------------------------------------------------------------+
bool FiboTimesLevelsSet(int              levels,           // количество линий уровня
                        double           &values[],        // значения линий уровня
                        color            &colors[],        // цвет линий уровня
                        ENUM_LINE_STYLE &styles[],         // стиль линий уровня
                        int              &widths[],        // толщина линий уровня
                        const long       chart_ID=0,       // ID графика
                        const string     name="FiboTimes") // имя объекта
  {
//--- проверим размеры массивов
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": длина массива не соответствуют количеству уровней, ошибка
      return(false);
     }
//--- установим количество уровней
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
   for(int i=0;i<levels;i++)
     {
      //--- значение уровня
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- цвет уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- стиль уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- толщина уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- описание уровня
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(values[i],1));
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Временных зон Фибоначчи"              |
//+------------------------------------------------------------------+
bool FiboTimesPointChange(const long   chart_ID=0,       // ID графика
                          const string name="FiboTimes", // имя объекта
                          const int    point_index=0,    // номер точки привязки
```

```
                           datetime     time=0,           // координата времени точки
                           double       price=0)          // координата цены точки при
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет "Временные зоны Фибоначчи"                               |
//+------------------------------------------------------------------+
bool FiboTimesDelete(const long   chart_ID=0,        // ID графика
                     const string name="FiboTimes") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим объект
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Временные зоны Фибоначчи\"! Код ошибки = ",GetLas
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Временных зон Фибоначчи", и   |
//| для пустых значений устанавливает значения по умолчанию          |
//+------------------------------------------------------------------+
void ChangeFiboTimesEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
  {
//--- если время первой точки не задано, то она будет на текущем баре
   if(!time1)
      time1=TimeCurrent();
```

```
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 2 бара левее второй
   if(!time2)
     {
      //--- массив для приема времени открытия 3 последних баров
      datetime temp[3];
      CopyTime(Symbol(),Period(),time1,3,temp);
      //--- установим первую точку на 2 бара левее второй
      time2=temp[0];
     }
//--- если цена второй точки не задана, то она совпадает с ценой первой точки
   if(!price2)
      price2=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки "Временных зон Фибоначчи"
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
```

```
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования "Временных зон Фибоначчи"
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- создадим объект
   if(!FiboTimesCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
   int h_steps=bars*2/5;
//--- перемещаем вторую точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d2<bars-1)
         d2+=1;
      //--- сдвигаем точку
      if(!FiboTimesPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   h_steps=bars*3/5;
//--- перемещаем первую точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
```

```
      if(d1<bars-1)
         d1+=1;
      //--- сдвигаем точку
      if(!FiboTimesPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим объект с графика
   FiboTimesDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_FIBOFAN

Веер Фибоначчи.

Примечание

Для "Веера Фибоначчи" можно указать количество линий-уровня, их значения и цвет.

Пример

Следующий скрипт создает и перемещает на графике "Веер Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Веер Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="FiboFan";          // Имя веера
input int             InpDate1=10;                // Дата 1-ой точки в %
input int             InpPrice1=25;               // Цена 1-ой точки в %
input int             InpDate2=30;                // Дата 2-ой точки в %
input int             InpPrice2=50;               // Цена 2-ой точки в %
input color           InpColor=clrRed;            // Цвет линии веера
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int             InpWidth=2;                 // Толщина линии
```

```
input bool             InpBack=false;              // Объект на заднем плане
input bool             InpSelection=true;          // Выделить для перемещений
input bool             InpHidden=true;             // Скрыт в списке объектов
input long             InpZOrder=0;                // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Веер Фибоначчи" по заданным координатам                 |
//+------------------------------------------------------------------+
bool FiboFanCreate(const long             chart_ID=0,        // ID графика
                   const string           name="FiboFan",    // имя веера
                   const int              sub_window=0,      // номер подокна
                   datetime               time1=0,           // время первой точки
                   double                 price1=0,          // цена первой точки
                   datetime               time2=0,           // время второй точки
                   double                 price2=0,          // цена второй точки
                   const color            clr=clrRed,        // цвет линии веера
                   const ENUM_LINE_STYLE  style=STYLE_SOLID, // стиль линии веера
                   const int              width=1,           // толщина линии веера
                   const bool             back=false,        // на заднем плане
                   const bool             selection=true,    // выделить для перемещени
                   const bool             hidden=true,       // скрыт в списке объекто
                   const long             z_order=0)         // приоритет на нажатие м
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeFiboFanEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Веер Фибоначчи" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOFAN,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Веер Фибоначчи\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения веера для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Задает количество уровней и их параметры                         |
//+------------------------------------------------------------------+
bool FiboFanLevelsSet(int             levels,         // количество линий уровня
                      double          &values[],      // значения линий уровня
                      color           &colors[],      // цвет линий уровня
                      ENUM_LINE_STYLE &styles[],      // стиль линий уровня
                      int             &widths[],      // толщина линий уровня
                      const long      chart_ID=0,     // ID графика
                      const string    name="FiboFan") // имя веера
  {
//--- проверим размеры массивов
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": длина массива не соответствуют количеству уровней, ошибка
      return(false);
     }
//--- установим количество уровней
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
   for(int i=0;i<levels;i++)
     {
      //--- значение уровня
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- цвет уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- стиль уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- толщина уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- описание уровня
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Веера Фибоначчи"                      |
//+------------------------------------------------------------------+
bool FiboFanPointChange(const long   chart_ID=0,     // ID графика
                        const string name="FiboFan", // имя веера
```

```
                        const int     point_index=0,  // номер точки привязки
                        datetime      time=0,         // координата времени точки прив
                        double        price=0)        // координата цены точки привязк
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет "Веер Фибоначчи"                                         |
//+------------------------------------------------------------------+
bool FiboFanDelete(const long   chart_ID=0,     // ID графика
                   const string name="FiboFan") // имя веера
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим веер
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Веер Фибоначчи\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Веера Фибоначчи", и для       |
//| пустых значений устанавливает значения по умолчанию              |
//+------------------------------------------------------------------+
void ChangeFiboFanEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
  {
//--- если время второй точки не задано, то она будет на текущем баре
   if(!time2)
```

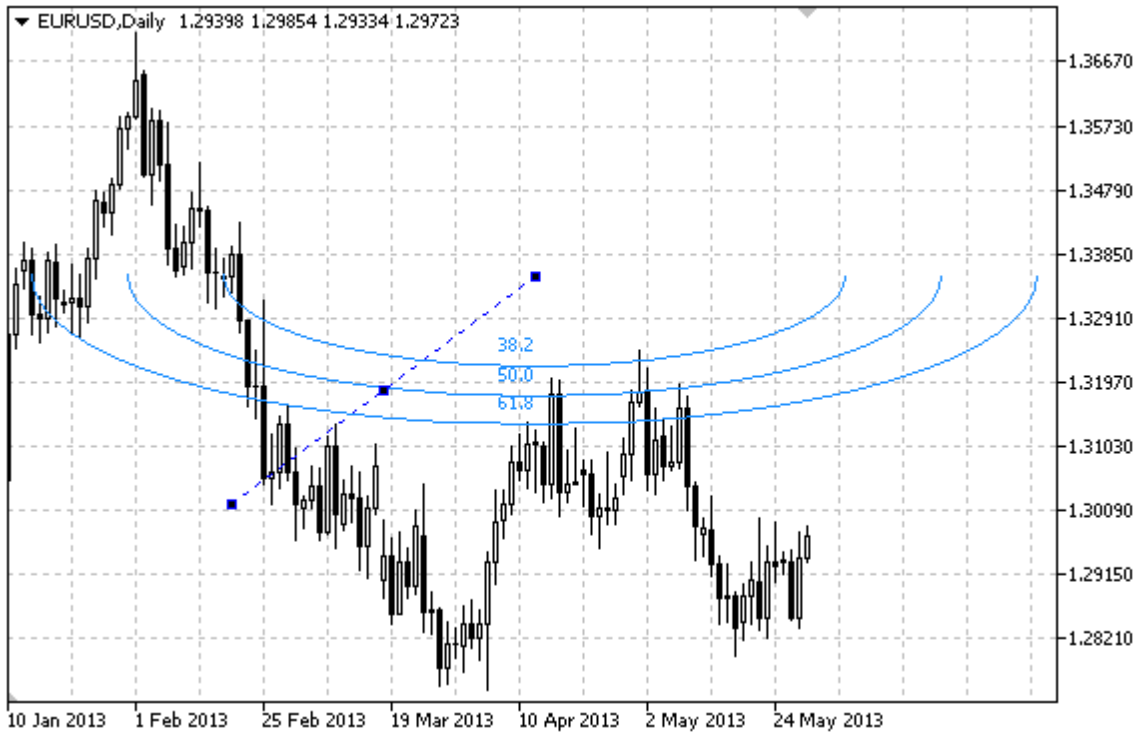
```
      time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то сдвинем ее на 200 пунктов ниже второй
   if(!price1)
      price1=price2-200*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки "Веера Фибоначчи"
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
```

```
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования "Веера Фибоначчи"
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- создадим объект
   if(!FiboFanCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки веера
//--- счетчик цикла
   int v_steps=accuracy/2;
//--- перемещаем первую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1<accuracy-1)
         p1+=1;
      //--- сдвигаем точку
      if(!FiboFanPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int h_steps=bars/4;
//--- перемещаем вторую точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d2<bars-1)
```

```
         d2+=1;
      //--- сдвигаем точку
      if(!FiboFanPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим объект с графика
   FiboFanDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_FIBOARC

Дуги Фибоначчи.



Примечание

Для "Дуг Фибоначчи" можно указать режим отображения всего эллипса целиком. Радиус кривизны линий можно задать изменяя масштаб и координаты точек привязки.

Также можно указать количество линий-уровня, их значения и цвет.

Пример

Следующий скрипт создает и перемещает на графике "Дуги Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Дуги Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="FiboArc";         // Имя объекта
input int             InpDate1=25;               // Дата 1-ой точки в %
input int             InpPrice1=25;              // Цена 1-ой точки в %
input int             InpDate2=35;               // Дата 2-ой точки в %
input int             InpPrice2=55;              // Цена 2-ой точки в %
input double          InpScale=3.0;              // Масштаб
```

```
input bool             InpFullEllipse=true;       // Форма дуг
input color            InpColor=clrRed;           // Цвет линии
input ENUM_LINE_STYLE  InpStyle=STYLE_DASHDOTDOT; // Стиль линии
input int              InpWidth=2;                // Толщина линии
input bool             InpBack=false;             // Объект на заднем плане
input bool             InpSelection=true;         // Выделить для перемещений
input bool             InpHidden=true;            // Скрыт в списке объектов
input long             InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Дуги Фибоначчи" по заданным координатам                 |
//+------------------------------------------------------------------+
bool FiboArcCreate(const long            chart_ID=0,          // ID графика
                   const string          name="FiboArc",      // имя объекта
                   const int             sub_window=0,        // номер подокна
                   datetime              time1=0,             // время первой точки
                   double                price1=0,            // цена первой точки
                   datetime              time2=0,             // время второй точки
                   double                price2=0,            // цена второй точки
                   const double          scale=1.0,           // масштаб
                   const bool            full_ellipse=false,  // форма дуг
                   const color           clr=clrRed,          // цвет линии
                   const ENUM_LINE_STYLE style=STYLE_SOLID,   // стиль линии
                   const int             width=1,             // толщина линии
                   const bool            back=false,          // на заднем плане
                   const bool            selection=true,      // выделить для перемеще
                   const bool            hidden=true,         // скрыт в списке объект
                   const long            z_order=0)           // приоритет на нажатие
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeFiboArcEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Дуги Фибоначчи" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOARC,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Дуги Фибоначчи\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим масштаб
   ObjectSetDouble(chart_ID,name,OBJPROP_SCALE,scale);
//--- установим отображение дуг в виде полного эллипса (true) или половины (false)
   ObjectSetInteger(chart_ID,name,OBJPROP_ELLIPSE,full_ellipse);
//--- установим цвет
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линии
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения дуг для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Задает количество уровней и их параметры                         |
//+------------------------------------------------------------------+
bool FiboArcLevelsSet(int              levels,         // количество линий уровня
                      double           &values[],      // значения линий уровня
                      color            &colors[],      // цвет линий уровня
                      ENUM_LINE_STYLE &styles[],       // стиль линий уровня
                      int              &widths[],      // толщина линий уровня
                      const long       chart_ID=0,     // ID графика
                      const string     name="FiboArc") // имя объекта
  {
//--- проверим размеры массивов
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": длина массива не соответсвуют количеству уровней, ошибка!
      return(false);
     }
//--- установим количество уровней
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
   for(int i=0;i<levels;i++)
     {
      //--- значение уровня
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- цвет уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- стиль уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- толщина уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- описание уровня
```

```
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Дуг Фибоначчи"                        |
//+------------------------------------------------------------------+
bool FiboArcPointChange(const long   chart_ID=0,     // ID графика
                        const string name="FiboArc", // имя объекта
                        const int    point_index=0,  // номер точки привязки
                        datetime     time=0,         // координата времени точки прив
                        double       price=0)        // координата цены точки привязк
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет "Дуги Фибоначчи"                                         |
//+------------------------------------------------------------------+
bool FiboArcDelete(const long   chart_ID=0,     // ID графика
                   const string name="FiboArc") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим объект
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Дуги Фибоначчи\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
```

```
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Дуг Фибоначчи", и для пустых  |
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeFiboArcEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2)
  {
//--- если время второй точки не задано, то она будет на текущем баре
   if(!time2)
      time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой точки не задано, то она лежит на 9 баров левее второй
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то сдвинем ее на 300 пунктов ниже второй
   if(!price1)
      price1=price2-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки "Дуг Фибоначчи"
   datetime date[];
   double   price[];
//--- выделение памяти
```

```
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования "Дуг Фибоначчи"
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- создадим объект
   if(!FiboArcCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpScale,
      InpFullEllipse,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOr
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
   int v_steps=accuracy/5;
//--- перемещаем первую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1<accuracy-1)
         p1+=1;
      //--- сдвигаем точку
      if(!FiboArcPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
```

```
      }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int h_steps=bars/5;
//--- перемещаем вторую точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d2<bars-1)
         d2+=1;
      //--- сдвигаем точку
      if(!FiboArcPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим объект с графика
   FiboArcDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_FIBOCHANNEL

Канал Фибоначчи.

Примечание

Для "Канала Фибоначчи" можно указать режим продолжения его отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно).

Также можно указать количество линий-уровня, их значения и цвет.

Пример

Следующий скрипт создает и перемещает на графике "Канал Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Канал Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="FiboChannel";      // Имя канала
input int             InpDate1=20;                // Дата 1-ой точки в %
input int             InpPrice1=10;               // Цена 1-ой точки в %
input int             InpDate2=60;                // Дата 2-ой точки в %
input int             InpPrice2=30;               // Цена 2-ой точки в %
input int             InpDate3=20;                // Дата 3-ей точки в %
```

```
input int              InpPrice3=25;              // Цена 3-ей точки в %
input color            InpColor=clrRed;           // Цвет канала
input ENUM_LINE_STYLE  InpStyle=STYLE_DASHDOTDOT; // Стиль линий канала
input int              InpWidth=2;                // Толщина линий канала
input bool             InpBack=false;             // Канал на заднем плане
input bool             InpSelection=true;         // Выделить для перемещений
input bool             InpRayLeft=false;          // Продолжение канала влево
input bool             InpRayRight=false;         // Продолжение канала вправо
input bool             InpHidden=true;            // Скрыт в списке объектов
input long             InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Канал Фибоначчи" по заданным координатам                |
//+------------------------------------------------------------------+
bool FiboChannelCreate(const long            chart_ID=0,          // ID графика
                       const string          name="FiboChannel", // имя канала
                       const int             sub_window=0,       // номер подокна
                       datetime              time1=0,            // время первой точки
                       double                price1=0,           // цена первой точки
                       datetime              time2=0,            // время второй точки
                       double                price2=0,           // цена второй точки
                       datetime              time3=0,            // время третьей точ
                       double                price3=0,           // цена третьей точки
                       const color           clr=clrRed,         // цвет канала
                       const ENUM_LINE_STYLE style=STYLE_SOLID,  // стиль линий канал
                       const int             width=1,            // толщина линий кан
                       const bool            back=false,         // на заднем плане
                       const bool            selection=true,     // выделить для пере
                       const bool            ray_left=false,     // продолжение канал
                       const bool            ray_right=false,    // продолжение канал
                       const bool            hidden=true,        // скрыт в списке об
                       const long            z_order=0)          // приоритет на нажа
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeFiboChannelEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим канал по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_FIBOCHANNEL,sub_window,time1,price1,time2,price
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Канал Фибоначчи\"! Код ошибки = ",GetLastError())
      return(false);
     }
//--- установим цвет канала
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий канала
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий канала
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения канала для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения канала влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения канала вправо
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Задает количество уровней и их параметры                         |
//+------------------------------------------------------------------+
bool FiboChannelLevelsSet(int             levels,             // количество линий уро
                          double          &values[],          // значения линий уровн
                          color           &colors[],          // цвет линий уровня
                          ENUM_LINE_STYLE &styles[],          // стиль линий уровня
                          int             &widths[],          // толщина линий уровня
                          const long      chart_ID=0,         // ID графика
                          const string    name="FiboChannel") // имя объекта
  {
//--- проверим размеры массивов
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": длина массива не соответсвуют количеству уровней, ошибка!
      return(false);
     }
//--- установим количество уровней
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
   for(int i=0;i<levels;i++)
     {
      //--- значение уровня
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- цвет уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- стиль уровня
```

```
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- толщина уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- описание уровня
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,DoubleToString(100*values[i],
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Канала Фибоначчи"                     |
//+------------------------------------------------------------------+
bool FiboChannelPointChange(const long   chart_ID=0,          // ID графика
                            const string name="FiboChannel", // имя канала
                            const int    point_index=0,       // номер точки привязки
                            datetime     time=0,              // координата времени то
                            double       price=0)             // координата цены точки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет канал                                                    |
//+------------------------------------------------------------------+
bool FiboChannelDelete(const long   chart_ID=0,         // ID графика
                       const string name="FiboChannel") // имя канала
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим канал
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Канал Фибоначчи\"! Код ошибки = ",GetLastError())
```

```
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Канала Фибоначчи", и для      |
//| пустых значений устанавливает значения по умолчанию              |
//+------------------------------------------------------------------+
void ChangeFiboChannelEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                  double &price2,datetime &time3,double &price3)
  {
//--- если время второй (правой) точки не задано, то она будет на текущем баре
   if(!time2)
      time2=TimeCurrent();
//--- если цена второй точки не задана, то она будет иметь значение Bid
   if(!price2)
      price2=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой (левой) точки не задано, то она лежит на 9 баров левее второй
   if(!time1)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time2,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то сдвинем ее на 300 пунктов выше второй
   if(!price1)
      price1=price2+300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время третьей точки не задано, то оно совпадает с временем первой точки
   if(!time3)
      time3=time1;
//--- если цена третьей точки не задана, то она совпадает с ценой второй точки
   if(!price3)
      price3=price2;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   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("Ошибка! Некорректные значения входных параметров!");
```

```
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки канала
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования канала
   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;
//--- создадим "Канал Фибоначчи"
   if(!FiboChannelCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],p
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidde
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки канала
//--- счетчик цикла
   int h_steps=bars/10;
//--- перемещаем первую точку привязки
```

```
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d1>1)
         d1-=1;
      //--- сдвигаем точку
      if(!FiboChannelPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int v_steps=accuracy/10;
//--- перемещаем вторую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p2>1)
         p2-=1;
      //--- сдвигаем точку
      if(!FiboChannelPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   v_steps=accuracy/15;
//--- перемещаем третью точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p3<accuracy-1)
         p3+=1;
      //--- сдвигаем точку
      if(!FiboChannelPointChange(0,InpName,2,date[d3],price[p3]))
         return;
```

```
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим канал с графика
   FiboChannelDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_EXPANSION

Расширение Фибоначчи.

Примечание

Для "Расширения Фибоначчи" можно указать режим продолжения отображения вправо и/или влево (свойства OBJPROP_RAY_RIGHT и OBJPROP_RAY_LEFT соответственно).

Также можно указать количество линий-уровня, их значения и цвет.

Пример

Следующий скрипт создает и перемещает на графике "Расширение Фибоначчи". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Расширение Фибоначчи\"."
#property description "Координаты точек привязки задаются в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="FiboExpansion";   // Имя объекта
input int             InpDate1=10;               // Дата 1-ой точки в %
input int             InpPrice1=55;              // Цена 1-ой точки в %
input int             InpDate2=30;               // Дата 2-ой точки в %
input int             InpPrice2=10;              // Цена 2-ой точки в %
input int             InpDate3=80;               // Дата 3-ей точки в %
```

```
input int              InpPrice3=75;              // Цена 3-ей точки в %
input color            InpColor=clrRed;           // Цвет объекта
input ENUM_LINE_STYLE  InpStyle=STYLE_DASHDOTDOT; // Стиль линий
input int              InpWidth=2;                // Толщина линий
input bool             InpBack=false;             // Объект на заднем плане
input bool             InpSelection=true;         // Выделить для перемещений
input bool             InpRayLeft=false;          // Продолжение объекта влево
input bool             InpRayRight=false;         // Продолжение объекта вправо
input bool             InpHidden=true;            // Скрыт в списке объектов
input long             InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Расширение Фибоначчи" по заданным координатам           |
//+------------------------------------------------------------------+
bool FiboExpansionCreate(const long            chart_ID=0,           // ID графика
                         const string          name="FiboExpansion", // имя канала
                         const int             sub_window=0,         // номер подокна
                         datetime              time1=0,              // время первой
                         double                price1=0,             // цена первой т
                         datetime              time2=0,              // время второй
                         double                price2=0,             // цена второй т
                         datetime              time3=0,              // время третьей
                         double                price3=0,             // цена третьей
                         const color           clr=clrRed,           // цвет объекта
                         const ENUM_LINE_STYLE style=STYLE_SOLID,    // стиль линий
                         const int             width=1,              // толщина линий
                         const bool            back=false,           // на заднем пла
                         const bool            selection=true,       // выделить для
                         const bool            ray_left=false,       // продолжение о
                         const bool            ray_right=false,      // продолжение о
                         const bool            hidden=true,          // скрыт в списк
                         const long            z_order=0)            // приоритет на
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeFiboExpansionEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Расширение Фибоначчи" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_EXPANSION,sub_window,time1,price1,time2,price2,
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Расширение Фибоначчи\"! Код ошибки = ",GetLastErr
      return(false);
     }
//--- установим цвет объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения объекта для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- включим (true) или отключим (false) режим продолжения отображения объекта влево
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_LEFT,ray_left);
//--- включим (true) или отключим (false) режим продолжения отображения объекта вправ
   ObjectSetInteger(chart_ID,name,OBJPROP_RAY_RIGHT,ray_right);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Задает количество уровней и их параметры                         |
//+------------------------------------------------------------------+
bool FiboExpansionLevelsSet(int             levels,               // количество линий
                            double          &values[],            // значения линий у
                            color           &colors[],            // цвет линий уровн
                            ENUM_LINE_STYLE &styles[],            // стиль линий уров
                            int             &widths[],            // толщина линий ур
                            const long      chart_ID=0,           // ID графика
                            const string    name="FiboExpansion") // имя объекта
  {
//--- проверим размеры массивов
   if(levels!=ArraySize(colors) || levels!=ArraySize(styles) ||
      levels!=ArraySize(widths) || levels!=ArraySize(widths))
     {
      Print(__FUNCTION__,": длина массива не соответсвуют количеству уровней, ошибка!
      return(false);
     }
//--- установим количество уровней
   ObjectSetInteger(chart_ID,name,OBJPROP_LEVELS,levels);
//--- установим свойства уровней в цикле
   for(int i=0;i<levels;i++)
     {
      //--- значение уровня
      ObjectSetDouble(chart_ID,name,OBJPROP_LEVELVALUE,i,values[i]);
      //--- цвет уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELCOLOR,i,colors[i]);
      //--- стиль уровня
```

```
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELSTYLE,i,styles[i]);
      //--- толщина уровня
      ObjectSetInteger(chart_ID,name,OBJPROP_LEVELWIDTH,i,widths[i]);
      //--- описание уровня
      ObjectSetString(chart_ID,name,OBJPROP_LEVELTEXT,i,"FE "+DoubleToString(100*valu
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Расширения Фибоначчи"                 |
//+------------------------------------------------------------------+
bool FiboExpansionPointChange(const long   chart_ID=0,           // ID графика
                              const string name="FiboExpansion", // имя объекта
                              const int    point_index=0,        // номер точки привя
                              datetime     time=0,               // координата времен
                              double       price=0)              // координата цены т
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет "Расширение Фибоначчи"                                   |
//+------------------------------------------------------------------+
bool FiboExpansionDelete(const long   chart_ID=0,           // ID графика
                         const string name="FiboExpansion") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим объект
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Расширение Фибоначчи\"! Код ошибки = ",GetLastErr
```

```
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Расширения Фибоначчи", и для  |
//| пустых значений устанавливает значения по умолчанию              |
//+------------------------------------------------------------------+
void ChangeFiboExpansionEmptyPoints(datetime &time1,double &price1,datetime &time2,
                                    double &price2,datetime &time3,double &price3)
  {
//--- если время третьей (правой) точки не задано, то она будет на текущем баре
   if(!time3)
      time3=TimeCurrent();
//--- если цена третьей точки не задана, то она будет иметь значение Bid
   if(!price3)
      price3=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время первой (левой) точки не задано, то она лежит на 9 баров левее третье
//--- массив для приема времени открытия 10 последних баров
   datetime temp[];
   ArrayResize(temp,10);
   if(!time1)
     {
      CopyTime(Symbol(),Period(),time3,10,temp);
      //--- установим первую точку на 9 баров левее второй
      time1=temp[0];
     }
//--- если цена первой точки не задана, то она совпадает с ценой третьей точки
   if(!price1)
      price1=price3;
//--- если время второй точки не задано, то она лежит на 7 баров левее третьей
   if(!time2)
      time2=temp[2];
//--- если цена второй точки не задана, сдвинем ее на 250 пунктов ниже первой
   if(!price2)
      price2=price1-250*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   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("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки объекта
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования "Расширения Фибоначчи"
   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;
//--- создадим "Расширение Фибоначчи"
   if(!FiboExpansionCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3]
      InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpRayLeft,InpRayRight,InpHidde
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
   int v_steps=accuracy/10;
```

```
//--- перемещаем первую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1>1)
         p1-=1;
      //--- сдвигаем точку
      if(!FiboExpansionPointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   v_steps=accuracy/2;
//--- перемещаем третью точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p3>1)
         p3-=1;
      //--- сдвигаем точку
      if(!FiboExpansionPointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   v_steps=accuracy*4/5;
//--- перемещаем вторую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p2<accuracy-1)
         p2+=1;
      //--- сдвигаем точку
      if(!FiboExpansionPointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
```

```
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим объект с графика
   FiboExpansionDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ELLIOTWAVE5

Импульсная волна Эллиота.

Примечание

Для "Импульсной волны Эллиота" можно включить/отключить режим соединения точек линиями (свойство OBJPROP_DRAWLINES), а также установить уровень волновой разметки (из перечисления ENUM_ELLIOT_WAVE_DEGREE).

Пример

Следующий скрипт создает и перемещает на графике "Импульсную волну Эллиота". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Импульсная волна Эллиота\".
#property description "Координаты точек привязки задаются в процентах от размеров"
#property description "окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string                InpName="ElliotWave5";   // Имя объекта
input int                   InpDate1=10;             // Дата 1-ой точки в %
input int                   InpPrice1=90;            // Цена 1-ой точки в %
input int                   InpDate2=20;             // Дата 2-ой точки в %
input int                   InpPrice2=40;            // Цена 2-ой точки в %
input int                   InpDate3=30;             // Дата 3-ей точки в %
```

```
input int                      InpPrice3=60;            // Цена 3-ей точки в %
input int                      InpDate4=40;             // Дата 4-ой точки в %
input int                      InpPrice4=10;            // Цена 4-ой точки в %
input int                      InpDate5=60;             // Дата 5-ой точки в %
input int                      InpPrice5=40;            // Цена 5-ой точки в %
input ENUM_ELLIOT_WAVE_DEGREE InpDegree=ELLIOTT_MINOR; // Уровень
input bool                     InpDrawLines=true;       // Отображение линий
input color                    InpColor=clrRed;         // Цвет линий
input ENUM_LINE_STYLE          InpStyle=STYLE_DASH;     // Стиль линий
input int                      InpWidth=2;              // Толщина линий
input bool                     InpBack=false;           // Объект на заднем плане
input bool                     InpSelection=true;       // Выделить для перемещений
input bool                     InpHidden=true;          // Скрыт в списке объектов
input long                     InpZOrder=0;             // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Импульсную волну Эллиота" по заданным координатам       |
//+------------------------------------------------------------------+
bool ElliotWave5Create(const long                  chart_ID=0,              // ID гр
                       const string                name="ElliotWave5",      // имя
                       const int                   sub_window=0,            // номер
                       datetime                    time1=0,                 // врем
                       double                      price1=0,                // цена
                       datetime                    time2=0,                 // врем
                       double                      price2=0,                // цена
                       datetime                    time3=0,                 // врем
                       double                      price3=0,                // цена
                       datetime                    time4=0,                 // врем
                       double                      price4=0,                // цена
                       datetime                    time5=0,                 // врем
                       double                      price5=0,                // цена
                       const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // степ
                       const bool                  draw_lines=true,         // отобр
                       const color                 clr=clrRed,              // цвет
                       const ENUM_LINE_STYLE       style=STYLE_SOLID,       // стил
                       const int                   width=1,                 // толщ
                       const bool                  back=false,              // на з
                       const bool                  selection=true,          // выде
                       const bool                  hidden=true,             // скры
                       const long                  z_order=0)               // прио
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeElliotWave5EmptyPoints(time1,price1,time2,price2,time3,price3,time4,price4,t
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Импульсную волну Эллиота" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE5,sub_window,time1,price1,time2,price
      price3,time4,price4,time5,price5))
     {
```

```
      Print(__FUNCTION__,
            ": не удалось создать \"Импульсную волну Эллиота\"! Код ошибки = ",GetLas
      return(false);
     }
//--- установим степень (размер волны)
   ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
//--- включим (true) или отключим (false) режим отображения линий
   ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
//--- установим цвет объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения объекта для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Импульсной волны Эллиота"             |
//+------------------------------------------------------------------+
bool ElliotWave5PointChange(const long   chart_ID=0,          // ID графика
                            const string name="ElliotWave5", // имя объекта
                            const int    point_index=0,      // номер точки привязки
                            datetime     time=0,             // координата времени то
                            double       price=0)            // координата цены точки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
```

```
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет "Импульсную волну Эллиота"                               |
//+------------------------------------------------------------------+
bool ElliotWave5Delete(const long   chart_ID=0,         // ID графика
                       const string name="ElliotWave5") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим объект
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Импульсную волну Эллиота\"! Код ошибки = ",GetLas
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Импульсной волны Эллиота", и  |
//| для пустых значений устанавливает значения по умолчанию          |
//+------------------------------------------------------------------+
void ChangeElliotWave5EmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2,
                                  datetime &time3,double &price3,
                                  datetime &time4,double &price4,
                                  datetime &time5,double &price5)
  {
//--- массив для приема времени открытия 10 последних баров
   datetime temp[];
   ArrayResize(temp,10);
//--- получим данные
   CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
//--- получим величину одного пункта на текущем графике
   double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время первой точки не задано, то она будет на 9 баре слева от последнего б
   if(!time1)
      time1=temp[0];
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
```

```
//--- если время второй точки не задано, то она будет на 7 баре слева от последнего б
   if(!time2)
      time2=temp[2];
//--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
   if(!price2)
      price2=price1-300*point;
//--- если время третьей точки не задано, то она будет на 5 баре слева от последнего
   if(!time3)
      time3=temp[4];
//--- если цена третьей точки не задана, сдвинем ее на 250 пунктов ниже первой
   if(!price3)
      price3=price1-250*point;
//--- если время четвертой точки не задано, то она будет на 3 баре слева от последнег
   if(!time4)
      time4=temp[6];
//--- если цена четвертой точки не задана, сдвинем ее на 550 пунктов ниже первой
   if(!price4)
      price4=price1-550*point;
//--- если время пятой точки не задано, то она будет на последнем баре
   if(!time5)
      time5=temp[9];
//--- если цена пятой точки не задана, сдвинем ее на 450 пунктов ниже первой
   if(!price5)
      price5=price1-450*point;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   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 ||
      InpDate4<0 || InpDate4>100 || InpPrice4<0 || InpPrice4>100 ||
      InpDate5<0 || InpDate5>100 || InpPrice5<0 || InpPrice5>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки объекта
   datetime date[];
   double   price[];
```

```
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования "Импульсной волны Эллиота"
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int d3=InpDate3*(bars-1)/100;
   int d4=InpDate4*(bars-1)/100;
   int d5=InpDate5*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
   int p3=InpPrice3*(accuracy-1)/100;
   int p4=InpPrice4*(accuracy-1)/100;
   int p5=InpPrice5*(accuracy-1)/100;
//--- создадим "Импульсную волну Эллиота"
   if(!ElliotWave5Create(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],p
      date[d4],price[p4],date[d5],price[p5],InpDegree,InpDrawLines,InpColor,InpStyle,
      InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
   int v_steps=accuracy/5;
//--- перемещаем пятую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p5<accuracy-1)
         p5+=1;
```

```
      //--- сдвигаем точку
      if(!ElliotWave5PointChange(0,InpName,4,date[d5],price[p5]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   v_steps=accuracy/5;
//--- перемещаем вторую и третью точки привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующие значения
      if(p2<accuracy-1)
         p2+=1;
      if(p3>1)
         p3-=1;
      //--- сдвигаем точки
      if(!ElliotWave5PointChange(0,InpName,1,date[d2],price[p2]))
         return;
      if(!ElliotWave5PointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   v_steps=accuracy*4/5;
//--- перемещаем первую и четвертую точки привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующие значения
      if(p1>1)
         p1-=1;
      if(p4<accuracy-1)
         p4+=1;
      //--- сдвигаем точки
      if(!ElliotWave5PointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!ElliotWave5PointChange(0,InpName,3,date[d4],price[p4]))
```

```
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим объект с графика
   ElliotWave5Delete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ELLIOTWAVE3

Корректирующая волна Эллиота.

Примечание

Для "Корректирующей волны Эллиота" можно включить/отключить режим соединения точек линиями (свойство OBJPROP_DRAWLINES), а также установить уровень волновой разметки (из перечисления ENUM_ELLIOT_WAVE_DEGREE).

Пример

Следующий скрипт создает и перемещает на графике "Корректирующую волну Эллиота". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Корректирующая волна Эллиот
#property description "Координаты точек привязки задаются в процентах от размеров окн
#property description "графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string                InpName="ElliotWave3";   // Имя объекта
input int                   InpDate1=10;             // Дата 1-ой точки в %
input int                   InpPrice1=90;            // Цена 1-ой точки в %
input int                   InpDate2=30;             // Дата 2-ой точки в %
input int                   InpPrice2=10;            // Цена 2-ой точки в %
input int                   InpDate3=50;             // Дата 3-ей точки в %
```

```
input int                      InpPrice3=40;             // Цена 3-ей точки в %
input ENUM_ELLIOT_WAVE_DEGREE  InpDegree=ELLIOTT_MINOR; // Уровень
input bool                     InpDrawLines=true;        // Отображение линий
input color                    InpColor=clrRed;          // Цвет линий
input ENUM_LINE_STYLE          InpStyle=STYLE_DASH;      // Стиль линий
input int                      InpWidth=2;               // Толщина линий
input bool                     InpBack=false;            // Объект на заднем плане
input bool                     InpSelection=true;        // Выделить для перемещений
input bool                     InpHidden=true;           // Скрыт в списке объектов
input long                     InpZOrder=0;              // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает "Корректирующую волну Эллиота" по заданным координатам   |
//+------------------------------------------------------------------+
bool ElliotWave3Create(const long                   chart_ID=0,               // ID гр
                       const string                 name="ElliotWave3",       // имя
                       const int                    sub_window=0,             // номе
                       datetime                     time1=0,                  // врем
                       double                       price1=0,                 // цена
                       datetime                     time2=0,                  // врем
                       double                       price2=0,                 // цена
                       datetime                     time3=0,                  // врем
                       double                       price3=0,                 // цена
                       const ENUM_ELLIOT_WAVE_DEGREE degree=ELLIOTT_MINUETTE, // степ
                       const bool                   draw_lines=true,          // отобр
                       const color                  clr=clrRed,               // цвет
                       const ENUM_LINE_STYLE        style=STYLE_SOLID,        // стил
                       const int                    width=1,                  // толщ
                       const bool                   back=false,               // на з
                       const bool                   selection=true,           // выде
                       const bool                   hidden=true,              // скры
                       const long                   z_order=0)                // прио
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeElliotWave3EmptyPoints(time1,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим "Корректирующую волну Эллиота" по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIOTWAVE3,sub_window,time1,price1,time2,price
     {
      Print(__FUNCTION__,
            ": не удалось создать \"Корректирующую волну Эллиота\"! Код ошибки = ",Ge
      return(false);
     }
//--- установим степень (размер волны)
   ObjectSetInteger(chart_ID,name,OBJPROP_DEGREE,degree);
//--- включим (true) или отключим (false) режим отображения линий
   ObjectSetInteger(chart_ID,name,OBJPROP_DRAWLINES,draw_lines);
//--- установим цвет объекта
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения объекта для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки "Корректирующей волны Эллиота"         |
//+------------------------------------------------------------------+
bool ElliotWave3PointChange(const long   chart_ID=0,          // ID графика
                            const string name="ElliotWave3", // имя объекта
                            const int    point_index=0,      // номер точки привязки
                            datetime     time=0,             // координата времени то
                            double       price=0)            // координата цены точки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет "Корректирующую волну Эллиота"                           |
```

```
//+------------------------------------------------------------------+
bool ElliotWave3Delete(const long   chart_ID=0,         // ID графика
                       const string name="ElliotWave3") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим объект
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить \"Корректирующую волну Эллиота\"! Код ошибки = ",Ge
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки "Корректирующей волны Эллиота",|
//| и для пустых значений устанавливает значения по умолчанию        |
//+------------------------------------------------------------------+
void ChangeElliotWave3EmptyPoints(datetime &time1,double &price1,
                                  datetime &time2,double &price2,
                                  datetime &time3,double &price3)
  {
//--- массив для приема времени открытия 10 последних баров
   datetime temp[];
   ArrayResize(temp,10);
//--- получим данные
   CopyTime(Symbol(),Period(),TimeCurrent(),10,temp);
//--- получим величину одного пункта на текущем графике
   double point=SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время первой точки не задано, то она будет на 9 баре слева от последнего б
   if(!time1)
      time1=temp[0];
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она будет на 5 баре слева от последнего б
   if(!time2)
      time2=temp[4];
//--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
   if(!price2)
      price2=price1-300*point;
//--- если время третьей точки не задано, то она будет на 1 баре слева от последнего
   if(!time3)
      time3=temp[8];
//--- если цена третьей точки не задана, сдвинем ее на 200 пунктов ниже первой
   if(!price3)
```

```
      price3=price1-200*point;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   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("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки объекта
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования "Корректирующей волны Эллиота"
   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;
```

```
//--- создадим "Корректирующую волну Эллиота"
   if(!ElliotWave3Create(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],p
      InpDegree,InpDrawLines,InpColor,InpStyle,InpWidth,InpBack,InpSelection,InpHidde
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки
//--- счетчик цикла
   int v_steps=accuracy/5;
//--- перемещаем третью точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p3<accuracy-1)
         p3+=1;
      //--- сдвигаем точку
      if(!ElliotWave3PointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   v_steps=accuracy*4/5;
//--- перемещаем первую и вторую точки привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующие значения
      if(p1>1)
         p1-=1;
      if(p2<accuracy-1)
         p2+=1;
      //--- сдвигаем точки
      if(!ElliotWave3PointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!ElliotWave3PointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
```

```
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим объект с графика
   ElliotWave3Delete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_RECTANGLE

Прямоугольник.

Примечание

Для прямоугольника можно установить режим заливки при помощи свойства OBJPROP_FILL.

Пример

Следующий скрипт создает и перемещает на графике прямоугольник. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит прямоугольник на графике."
#property description "Координаты точек привязки задаются в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="Rectangle"; // Имя прямоугольника
input int             InpDate1=40;         // Дата 1-ой точки в %
input int             InpPrice1=40;        // Цена 1-ой точки в %
input int             InpDate2=60;         // Дата 2-ой точки в %
input int             InpPrice2=60;        // Цена 2-ой точки в %
input color           InpColor=clrRed;     // Цвет прямоугольника
input ENUM_LINE_STYLE InpStyle=STYLE_DASH; // Стиль линий прямоугольника
input int             InpWidth=2;          // Толщина линий прямоугольника
```

```
input bool              InpFill=true;         // Заливка прямоугольника цветом
input bool              InpBack=false;        // Прямоугольника на заднем плане
input bool              InpSelection=true;    // Выделить для перемещений
input bool              InpHidden=true;       // Скрыт в списке объектов
input long              InpZOrder=0;          // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает прямоугольник по заданным координатам                    |
//+------------------------------------------------------------------+
bool RectangleCreate(const long            chart_ID=0,        // ID графика
                     const string          name="Rectangle",  // имя прямоугольника
                     const int             sub_window=0,      // номер подокна
                     datetime              time1=0,           // время первой точки
                     double                price1=0,          // цена первой точки
                     datetime              time2=0,           // время второй точки
                     double                price2=0,          // цена второй точки
                     const color           clr=clrRed,        // цвет прямоугольника
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий прямоуго
                     const int             width=1,           // толщина линий прямоу
                     const bool            fill=false,        // заливка прямоугольни
                     const bool            back=false,        // на заднем плане
                     const bool            selection=true,    // выделить для перемещ
                     const bool            hidden=true,       // скрыт в списке объек
                     const long            z_order=0)         // приоритет на нажатие
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeRectangleEmptyPoints(time1,price1,time2,price2);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим прямоугольник по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE,sub_window,time1,price1,time2,price2))
     {
      Print(__FUNCTION__,
            ": не удалось создать прямоугольник! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет прямоугольника
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий прямоугольника
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий прямоугольника
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки прямоугольника
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения прямоугольника для перемеще
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
```

```
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки прямоугольника                         |
//+------------------------------------------------------------------+
bool RectanglePointChange(const long   chart_ID=0,       // ID графика
                          const string name="Rectangle", // имя прямоугольника
                          const int    point_index=0,    // номер точки привязки
                          datetime     time=0,           // координата времени точки
                          double       price=0)          // координата цены точки при
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет прямоугольник                                            |
//+------------------------------------------------------------------+
bool RectangleDelete(const long   chart_ID=0,       // ID графика
                     const string name="Rectangle") // имя прямоугольника
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим прямоугольник
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
```

```
            ": не удалось удалить прямоугольник! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки прямоугольника, и для пустых   |
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeRectangleEmptyPoints(datetime &time1,double &price1,
                                datetime &time2,double &price2)
  {
//--- если время первой точки не задано, то она будет на текущем баре
   if(!time1)
      time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее второй
   if(!time2)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- установим вторую точку на 9 баров левее первой
      time2=temp[0];
     }
//--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate1<0 || InpDate1>100 || InpPrice1<0 || InpPrice1>100 ||
      InpDate2<0 || InpDate2>100 || InpPrice2<0 || InpPrice2>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
```

```
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки прямоугольника
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования прямоугольника
   int d1=InpDate1*(bars-1)/100;
   int d2=InpDate2*(bars-1)/100;
   int p1=InpPrice1*(accuracy-1)/100;
   int p2=InpPrice2*(accuracy-1)/100;
//--- создадим прямоугольник
   if(!RectangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],InpColor,
      InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки прямоугольника
//--- счетчик цикла
   int h_steps=bars/2;
//--- перемещаем точки привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующие значения
      if(d1<bars-1)
         d1+=1;
      if(d2>1)
         d2-=1;
      //--- сдвигаем точки
```

```
      if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int v_steps=accuracy/2;
//--- перемещаем точки привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующие значения
      if(p1<accuracy-1)
         p1+=1;
      if(p2>1)
         p2-=1;
      //--- сдвигаем точки
      if(!RectanglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!RectanglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим прямоугольник с графика
   RectangleDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_TRIANGLE

Треугольник.

Примечание

Для треугольника можно установить режим заливки при помощи свойства OBJPROP_FILL.

Пример

Следующий скрипт создает и перемещает на графике треугольник. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит треугольник на графике."
#property description "Координаты точек привязки задаются в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="Triangle";        // Имя треугольника
input int              InpDate1=25;               // Дата 1-ой точки в %
input int              InpPrice1=50;              // Цена 1-ой точки в %
input int              InpDate2=70;               // Дата 2-ой точки в %
input int              InpPrice2=70;              // Цена 2-ой точки в %
input int              InpDate3=65;               // Дата 3-ей точки в %
input int              InpPrice3=20;              // Цена 3-ей точки в %
input color            InpColor=clrRed;           // Цвет треугольника
```

```
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий треугольника
input int             InpWidth=2;                // Толщина линий треугольника
input bool            InpFill=false;             // Заливка треугольника цветом
input bool            InpBack=false;             // Треугольника на заднем плане
input bool            InpSelection=true;         // Выделить для перемещений
input bool            InpHidden=true;            // Скрыт в списке объектов
input long            InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает треугольник по заданным координатам                      |
//+------------------------------------------------------------------+
bool TriangleCreate(const long            chart_ID=0,        // ID графика
                    const string          name="Triangle",   // имя треугольника
                    const int             sub_window=0,      // номер подокна
                    datetime              time1=0,           // время первой точки
                    double                price1=0,          // цена первой точки
                    datetime              time2=0,           // время второй точки
                    double                price2=0,          // цена второй точки
                    datetime              time3=0,           // время третьей точки
                    double                price3=0,          // цена третьей точки
                    const color           clr=clrRed,        // цвет треугольника
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий треугольни
                    const int             width=1,           // толщина линий треугол
                    const bool            fill=false,        // заливка треугольника
                    const bool            back=false,        // на заднем плане
                    const bool            selection=true,    // выделить для перемеще
                    const bool            hidden=true,       // скрыт в списке объект
                    const long            z_order=0)         // приоритет на нажатие
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeTriangleEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим треугольник по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_TRIANGLE,sub_window,time1,price1,time2,price2,t
     {
      Print(__FUNCTION__,
            ": не удалось создать треугольник! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет треугольника
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий треугольника
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий треугольника
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки треугольника
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим выделения треугольник для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки треугольника                           |
//+------------------------------------------------------------------+
bool TrianglePointChange(const long   chart_ID=0,       // ID графика
                         const string name="Triangle", // имя треугольника
                         const int    point_index=0,   // номер точки привязки
                         datetime     time=0,          // координата времени точки при
                         double       price=0)         // координата цены точки привя
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет треугольник                                              |
//+------------------------------------------------------------------+
bool TriangleDelete(const long   chart_ID=0,       // ID графика
                    const string name="Triangle") // имя треугольника
  {
//--- сбросим значение ошибки
   ResetLastError();
```

```
//--- удалим треугольник
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить треугольник! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки треугольника, и для пустых     |
//| значений устанавливает значения по умолчанию                     |
//+------------------------------------------------------------------+
void ChangeTriangleEmptyPoints(datetime &time1,double &price1,
                               datetime &time2,double &price2,
                               datetime &time3,double &price3)
  {
//--- если время первой точки не задано, то она будет на текущем баре
   if(!time1)
      time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее второй
   if(!time2)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- установим вторую точку на 9 баров левее первой
      time2=temp[0];
     }
//--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время третьей точки не задано, то оно будет совпадать с датой второй точки
   if(!time3)
      time3=time2;
//--- если цена первой точки не задана, то она будет совпадать с ценой первой точки
   if(!price3)
      price3=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
```

```
//--- проверим входные параметры на корректность
   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("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки треугольника
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования треугольника
   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;
//--- создадим треугольник
   if(!TriangleCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],pric
      InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
```

```
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точки привязки треугольника
//--- счетчик цикла
   int v_steps=accuracy*3/10;
//--- перемещаем первую точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p1>1)
         p1-=1;
      //--- сдвигаем точку
      if(!TrianglePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int h_steps=bars*9/20-1;
//--- перемещаем вторую точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d2>1)
         d2-=1;
      //--- сдвигаем точку
      if(!TrianglePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   v_steps=accuracy/4;
//--- перемещаем третью точку привязки
   for(int i=0;i<v_steps;i++)
     {
```

```
      //--- возьмем следующее значение
      if(p3<accuracy-1)
         p3+=1;
      //--- сдвигаем точку
      if(!TrianglePointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим треугольник с графика
   TriangleDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ELLIPSE

Эллипс.

Примечание

Для эллипса можно установить режим заливки при помощи свойства OBJPROP_FILL.

Пример

Следующий скрипт создает и перемещает на графике эллипс. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит эллипс на графике."
#property description "Координаты точек привязки задаются"
#property description "в процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpName="Ellipse";         // Имя эллипса
input int             InpDate1=30;               // Дата 1-ой точки в %
input int             InpPrice1=20;              // Цена 1-ой точки в %
input int             InpDate2=70;               // Дата 2-ой точки в %
input int             InpPrice2=80;              // Цена 2-ой точки в %
input int             InpDate3=50;               // Дата 3-ей точки в %
input int             InpPrice3=60;              // Цена 3-ей точки в %
input color           InpColor=clrRed;           // Цвет эллипса
input ENUM_LINE_STYLE InpStyle=STYLE_DASHDOTDOT; // Стиль линий эллипса
```

```
input int              InpWidth=2;                // Толщина линий эллипса
input bool             InpFill=false;             // Заливка эллипса цветом
input bool             InpBack=false;             // Эллипс на заднем плане
input bool             InpSelection=true;         // Выделить для перемещений
input bool             InpHidden=true;            // Скрыт в списке объектов
input long             InpZOrder=0;               // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает эллипс по заданным координатам                           |
//+------------------------------------------------------------------+
bool EllipseCreate(const long            chart_ID=0,        // ID графика
                   const string          name="Ellipse",    // имя эллипса
                   const int             sub_window=0,      // номер подокна
                   datetime              time1=0,           // время первой точки
                   double                price1=0,          // цена первой точки
                   datetime              time2=0,           // время второй точки
                   double                price2=0,          // цена второй точки
                   datetime              time3=0,           // время третьей точки
                   double                price3=0,          // цена третьей точки
                   const color           clr=clrRed,        // цвет эллипса
                   const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линий эллипса
                   const int             width=1,           // толщина линий эллипса
                   const bool            fill=false,        // заливка эллипса цветом
                   const bool            back=false,        // на заднем плане
                   const bool            selection=true,    // выделить для перемещени
                   const bool            hidden=true,       // скрыт в списке объекто
                   const long            z_order=0)         // приоритет на нажатие м
  {
//--- установим координаты точек привязки, если они не заданы
   ChangeEllipseEmptyPoints(time1,price1,time2,price2,time3,price3);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим эллипс по заданным координатам
   if(!ObjectCreate(chart_ID,name,OBJ_ELLIPSE,sub_window,time1,price1,time2,price2,ti
     {
      Print(__FUNCTION__,
            ": не удалось создать эллипс! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет эллипса
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линий эллипса
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину линий эллипса
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- включим (true) или отключим (false) режим заливки эллипса
   ObjectSetInteger(chart_ID,name,OBJPROP_FILL,fill);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
```

```
//--- включим (true) или отключим (false) режим выделения эллипс для перемещений
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки эллипса                                |
//+------------------------------------------------------------------+
bool EllipsePointChange(const long   chart_ID=0,     // ID графика
                        const string name="Ellipse", // имя эллипса
                        const int    point_index=0,  // номер точки привязки
                        datetime     time=0,         // координата времени точки прив
                        double       price=0)        // координата цены точки привязк
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,point_index,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет эллипс                                                   |
//+------------------------------------------------------------------+
bool EllipseDelete(const long   chart_ID=0,     // ID графика
                   const string name="Ellipse") // имя эллипса
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим эллипс
```

```
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить эллипс! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точек привязки эллипса, и для пустых значений |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeEllipseEmptyPoints(datetime &time1,double &price1,
                              datetime &time2,double &price2,
                              datetime &time3,double &price3)
  {
//--- если время первой точки не задано, то она будет на текущем баре
   if(!time1)
      time1=TimeCurrent();
//--- если цена первой точки не задана, то она будет иметь значение Bid
   if(!price1)
      price1=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- если время второй точки не задано, то она лежит на 9 баров левее второй
   if(!time2)
     {
      //--- массив для приема времени открытия 10 последних баров
      datetime temp[10];
      CopyTime(Symbol(),Period(),time1,10,temp);
      //--- установим вторую точку на 9 баров левее первой
      time2=temp[0];
     }
//--- если цена второй точки не задана, сдвинем ее на 300 пунктов ниже первой
   if(!price2)
      price2=price1-300*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//--- если время третьей точки не задано, то оно будет совпадать с датой второй точки
   if(!time3)
      time3=time2;
//--- если цена третьей точки не задана, то она будет совпадать с ценой первой точки
   if(!price3)
      price3=price1;
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
```

```
   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("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точек привязки эллипса
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования эллипса
   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;
//--- создадим эллипс
   if(!EllipseCreate(0,InpName,0,date[d1],price[p1],date[d2],price[p2],date[d3],price
      InpColor,InpStyle,InpWidth,InpFill,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
```

```
   Sleep(1000);
//--- теперь будем перемещать точки привязки эллипса
//--- счетчик цикла
   int v_steps=accuracy/5;
//--- перемещаем первую и вторую точки привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующие значения
      if(p1<accuracy-1)
         p1+=1;
      if(p2>1)
         p2-=1;
      //--- сдвигаем точки
      if(!EllipsePointChange(0,InpName,0,date[d1],price[p1]))
         return;
      if(!EllipsePointChange(0,InpName,1,date[d2],price[p2]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int h_steps=bars/5;
//--- перемещаем третью точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d3>1)
         d3-=1;
      //--- сдвигаем точку
      if(!EllipsePointChange(0,InpName,2,date[d3],price[p3]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим эллипс с графика
   EllipseDelete(0,InpName);
```

```
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROW_THUMB_UP

Знак "Хорошо".

Примечание

Положение точки привязки относительно знака можно выбрать из перечисления ENUM_ARROW_ANCHOR.

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства OBJPROP_WIDTH при написании кода в MetaEditor.

Пример

Следующий скрипт создает и перемещает на графике знак "Хорошо". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует знак \"Хорошо\" (большой палец вверх)."
#property description "Координата точки привязки задается в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string             InpName="ThumbUp";     // Имя знака
input int                InpDate=75;            // Дата точки привязки в %
input int                InpPrice=25;           // Цена точки привязки в %
input ENUM_ARROW_ANCHOR  InpAnchor=ANCHOR_TOP;  // Способ привязки
input color              InpColor=clrRed;       // Цвет знака
```

```
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;   // Стиль окаймляющей линии
input int               InpWidth=5;           // Размер знака
input bool              InpBack=false;        // Знак на заднем плане
input bool              InpSelection=true;    // Выделить для перемещений
input bool              InpHidden=true;       // Скрыт в списке объектов
input long              InpZOrder=0;          // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает знак "Хорошо"                                            |
//+------------------------------------------------------------------+
bool ArrowThumbUpCreate(const long              chart_ID=0,           // ID графика
                        const string            name="ThumbUp",       // имя знака
                        const int               sub_window=0,         // номер подокна
                        datetime                time=0,               // время точки
                        double                  price=0,              // цена точки пр
                        const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привяз
                        const color             clr=clrRed,           // цвет знака
                        const ENUM_LINE_STYLE   style=STYLE_SOLID,    // стиль окаймл
                        const int               width=3,              // размер знака
                        const bool              back=false,           // на заднем пл
                        const bool              selection=true,       // выделить для
                        const bool              hidden=true,          // скрыт в спис
                        const long              z_order=0)            // приоритет на
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим знак
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_UP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать знак \"Хорошо\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowThumbUpMove(const long   chart_ID=0,      // ID графика
                      const string name="ThumbUp", // имя объекта
                      datetime     time=0,         // координата времени точки привяз
                      double       price=0)        // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Меняет способ привязки знака "Хорошо"                            |
//+------------------------------------------------------------------+
bool ArrowThumbUpAnchorChange(const long              chart_ID=0,        // ID график
                              const string            name="ThumbUp",    // имя объек
                              const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ пр
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим способ привязки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
```

```
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет знак "Хорошо"                                            |
//+------------------------------------------------------------------+
bool ArrowThumbUpDelete(const long   chart_ID=0,     // ID графика
                        const string name="ThumbUp") // имя знака
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим знак
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить знак \"Хорошо\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
```

```
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования знака
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- создадим знак "Хорошо" на графике
   if(!ArrowThumbUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
   int h_steps=bars/4;
//--- перемещаем точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d>1)
         d-=1;
      //--- сдвигаем точку
      if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
```

```
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int v_steps=accuracy/4;
//--- перемещаем точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p<accuracy-1)
         p+=1;
      //--- сдвигаем точку
      if(!ArrowThumbUpMove(0,InpName,date[d],price[p]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- меняем положение точки привязки относительно знака
   ArrowThumbUpAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- перерисуем график
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим знак с графика
   ArrowThumbUpDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROW_THUMB_DOWN

Знак "Плохо".

Примечание

Положение точки привязки относительно знака можно выбрать из перечисления ENUM_ARROW_ANCHOR.

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства OBJPROP_WIDTH при написании кода в MetaEditor.

Пример

Следующий скрипт создает и перемещает на графике знак "Плохо". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует знак \"Плохо\" (большой палец вниз)."
#property description "Координата точки привязки задается в процентах от"
#property description "размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string            InpName="ThumbDown";      // Имя знака
input int               InpDate=25;               // Дата точки привязки в %
input int               InpPrice=75;              // Цена точки привязки в %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Способ привязки
```

```
input color             InpColor=clrRed;         // Цвет знака
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;      // Стиль окаймляющей линии
input int               InpWidth=5;              // Размер знака
input bool              InpBack=false;           // Знак на заднем плане
input bool              InpSelection=true;       // Выделить для перемещений
input bool              InpHidden=true;          // Скрыт в списке объектов
input long              InpZOrder=0;             // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает знак "Плохо"                                             |
//+------------------------------------------------------------------+
bool ArrowThumbDownCreate(const long              chart_ID=0,           // ID графика
                          const string            name="ThumbDown",     // имя знака
                          const int               sub_window=0,         // номер подо
                          datetime                time=0,               // время точки
                          double                  price=0,              // цена точки
                          const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ при
                          const color             clr=clrRed,           // цвет знака
                          const ENUM_LINE_STYLE   style=STYLE_SOLID,    // стиль окай
                          const int               width=3,              // размер зна
                          const bool              back=false,           // на заднем
                          const bool              selection=true,       // выделить д
                          const bool              hidden=true,          // скрыт в сп
                          const long              z_order=0)            // приоритет
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим знак
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_THUMB_DOWN,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать знак \"Плохо\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
```

```
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowThumbDownMove(const long   chart_ID=0,        // ID графика
                        const string name="ThumbDown", // имя объекта
                        datetime     time=0,           // координата времени точки при
                        double       price=0)          // координата цены точки привя
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Меняет способ привязки знака "Плохо"                             |
//+------------------------------------------------------------------+
bool ArrowThumbDownAnchorChange(const long              chart_ID=0,        // ID граф
                                const string            name="ThumbDown",  // имя объ
                                const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим способ привязки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
```

```
            ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет знак "Плохо"                                             |
//+------------------------------------------------------------------+
bool ArrowThumbDownDelete(const long   chart_ID=0,       // ID графика
                          const string name="ThumbDown") // имя знака
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим знак
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить знак \"Плохо\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
```

```
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования знака
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- создадим знак "Плохо" на графике
   if(!ArrowThumbDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
   int h_steps=bars/4;
//--- перемещаем точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d<bars-1)
         d+=1;
      //--- сдвигаем точку
```

```
      if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- счетчик цикла
   int v_steps=accuracy/4;
//--- перемещаем точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p>1)
         p-=1;
      //--- сдвигаем точку
      if(!ArrowThumbDownMove(0,InpName,date[d],price[p]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- меняем положение точки привязки относительно знака
   ArrowThumbDownAnchorChange(0,InpName,ANCHOR_TOP);
//--- перерисуем график
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим знак с графика
   ArrowThumbDownDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROW_UP

Знак "Стрелка вверх".

Примечание

Положение точки привязки относительно знака можно выбрать из перечисления ENUM_ARROW_ANCHOR.

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства OBJPROP_WIDTH при написании кода в MetaEditor.

Пример

Следующий скрипт создает и перемещает на графике знак "Стрелка вверх". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует знак \"Стрелка вверх\"."
#property description "Координата точки привязки задается в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string            InpName="ArrowUp";     // Имя знака
input int               InpDate=25;            // Дата точки привязки в %
input int               InpPrice=25;           // Цена точки привязки в %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP; // Способ привязки
input color             InpColor=clrRed;       // Цвет знака
```

```
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;   // Стиль окаймляющей линии
input int               InpWidth=5;           // Размер знака
input bool              InpBack=false;        // Знак на заднем плане
input bool              InpSelection=false;   // Выделить для перемещений
input bool              InpHidden=true;       // Скрыт в списке объектов
input long              InpZOrder=0;          // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает знак "Стрелка вверх"                                     |
//+------------------------------------------------------------------+
bool ArrowUpCreate(const long              chart_ID=0,           // ID графика
                   const string            name="ArrowUp",       // имя знака
                   const int               sub_window=0,         // номер подокна
                   datetime                time=0,               // время точки привя
                   double                  price=0,              // цена точки привяз
                   const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязки
                   const color             clr=clrRed,           // цвет знака
                   const ENUM_LINE_STYLE   style=STYLE_SOLID,    // стиль окаймляющей
                   const int               width=3,              // размер знака
                   const bool              back=false,           // на заднем плане
                   const bool              selection=true,       // выделить для пере
                   const bool              hidden=true,          // скрыт в списке об
                   const long              z_order=0)            // приоритет на нажа
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим знак
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_UP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать знак \"Стрелка вверх\"! Код ошибки = ",GetLastError
      return(false);
     }
//--- установим способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowUpMove(const long   chart_ID=0,     // ID графика
                 const string name="ArrowUp", // имя объекта
                 datetime     time=0,         // координата времени точки привязки
                 double       price=0)        // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Меняет способ привязки знака "Стрелка вверх"                     |
//+------------------------------------------------------------------+
bool ArrowUpAnchorChange(const long              chart_ID=0,        // ID графика
                         const string            name="ArrowUp",    // имя объекта
                         const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ привязки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим положение точки привязки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
```

```
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет знак "Стрелка вверх"                                     |
//+------------------------------------------------------------------+
bool ArrowUpDelete(const long   chart_ID=0,     // ID графика
                   const string name="ArrowUp") // имя знака
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим знак
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить знак \"Стрелка вверх\"! Код ошибки = ",GetLastError
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
```

```
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования знака
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- создадим знак "Стрелка вверх" на графике
   if(!ArrowUpCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
   int v_steps=accuracy/2;
//--- перемещаем точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p<accuracy-1)
         p+=1;
      //--- сдвигаем точку
      if(!ArrowUpMove(0,InpName,date[d],price[p]))
```

```
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- меняем положение точки привязки относительно знака
   ArrowUpAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- перерисуем график
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим знак с графика
   ArrowUpDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROW_DOWN

Знак "Стрелка вниз".

Примечание

Положение точки привязки относительно знака можно выбрать из перечисления ENUM_ARROW_ANCHOR.

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства OBJPROP_WIDTH при написании кода в MetaEditor.

Пример

Следующий скрипт создает и перемещает на графике знак "Стрелка вниз". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует знак \"Стрелка вниз\"."
#property description "Координата точки привязки задается в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string            InpName="ArrowDown";      // Имя знака
input int               InpDate=75;               // Дата точки привязки в %
input int               InpPrice=75;              // Цена точки привязки в %
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_BOTTOM; // Способ привязки
input color             InpColor=clrRed;          // Цвет знака
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;       // Стиль окаймляющей линии
```

```
input int                InpWidth=5;               // Размер знака
input bool               InpBack=false;            // Знак на заднем плане
input bool               InpSelection=false;       // Выделить для перемещений
input bool               InpHidden=true;           // Скрыт в списке объектов
input long               InpZOrder=0;              // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает знак "Стрелка вниз"                                      |
//+------------------------------------------------------------------+
bool ArrowDownCreate(const long              chart_ID=0,           // ID графика
                     const string            name="ArrowDown",     // имя знака
                     const int               sub_window=0,         // номер подокна
                     datetime                time=0,               // время точки при
                     double                  price=0,              // цена точки прив
                     const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязки
                     const color             clr=clrRed,           // цвет знака
                     const ENUM_LINE_STYLE   style=STYLE_SOLID,    // стиль окаймляющ
                     const int               width=3,              // размер знака
                     const bool              back=false,           // на заднем плане
                     const bool              selection=true,       // выделить для пер
                     const bool              hidden=true,          // скрыт в списке
                     const long              z_order=0)            // приоритет на на
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим знак
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_DOWN,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать знак \"Стрелка вниз\"! Код ошибки = ",GetLastError(
      return(false);
     }
//--- способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowDownMove(const long   chart_ID=0,        // ID графика
                   const string name="ArrowDown", // имя объекта
                   datetime     time=0,           // координата времени точки привязки
                   double       price=0)          // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Меняет способ привязки знака "Стрелка вниз"                      |
//+------------------------------------------------------------------+
bool ArrowDownAnchorChange(const long              chart_ID=0,        // ID графика
                           const string            name="ArrowDown",  // имя объекта
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ привя
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим положение точки привязки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
      return(false);
```

```
      }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет знак "Стрелка вниз"                                      |
//+------------------------------------------------------------------+
bool ArrowDownDelete(const long   chart_ID=0,       // ID графика
                     const string name="ArrowDown") // имя знака
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим знак
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить знак \"Стрелка вниз\"! Код ошибки = ",GetLastError(
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
```

```
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования знака
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- создадим знак "Стрелка вниз" на графике
   if(!ArrowDownCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
   int v_steps=accuracy/2;
//--- перемещаем точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p>1)
         p-=1;
      //--- сдвигаем точку
      if(!ArrowDownMove(0,InpName,date[d],price[p]))
         return;
```

```
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- меняем положение точки привязки относительно знака
   ArrowDownAnchorChange(0,InpName,ANCHOR_TOP);
//--- перерисуем график
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим знак с графика
   ArrowDownDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROW_STOP

Знак "Стоп".

Примечание

Положение точки привязки относительно знака можно выбрать из перечисления ENUM_ARROW_ANCHOR.

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства OBJPROP_WIDTH при написании кода в MetaEditor.

Пример

Следующий скрипт создает и перемещает на графике знак "Стоп". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует знак \"Стоп\"."
#property description "Координата точки привязки задается в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string             InpName="ArrowStop";      // Имя знака
input int                InpDate=10;               // Дата точки привязки в %
input int                InpPrice=50;              // Цена точки привязки в %
input ENUM_ARROW_ANCHOR  InpAnchor=ANCHOR_BOTTOM;  // Способ привязки
input color              InpColor=clrRed;          // Цвет знака
```

```
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;      // Стиль окаймляющей линии
input int               InpWidth=5;              // Размер знака
input bool              InpBack=false;           // Знак на заднем плане
input bool              InpSelection=false;      // Выделить для перемещений
input bool              InpHidden=true;          // Скрыт в списке объектов
input long              InpZOrder=0;             // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает знак "Стоп"                                              |
//+------------------------------------------------------------------+
bool ArrowStopCreate(const long              chart_ID=0,           // ID графика
                     const string            name="ArrowStop",     // имя знака
                     const int               sub_window=0,         // номер подокна
                     datetime                time=0,               // время точки при
                     double                  price=0,              // цена точки прив
                     const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязки
                     const color             clr=clrRed,           // цвет знака
                     const ENUM_LINE_STYLE   style=STYLE_SOLID,    // стиль окаймляющ
                     const int               width=3,              // размер знака
                     const bool              back=false,           // на заднем плане
                     const bool              selection=true,       // выделить для пе
                     const bool              hidden=true,          // скрыт в списке
                     const long              z_order=0)            // приоритет на на
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим знак
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_STOP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать знак \"Стоп\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowStopMove(const long   chart_ID=0,        // ID графика
                   const string name="ArrowStop", // имя объекта
                   datetime     time=0,           // координата времени точки привязки
                   double       price=0)          // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Меняет способ привязки знака "Стоп"                              |
//+------------------------------------------------------------------+
bool ArrowStopAnchorChange(const long              chart_ID=0,         // ID графика
                           const string            name="ArrowStop",   // имя объекта
                           const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // положение то
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим способ привязки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
```

```
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет знак "Стоп"                                              |
//+------------------------------------------------------------------+
bool ArrowStopDelete(const long   chart_ID=0,       // ID графика
                     const string name="ArrowStop") // имя знака
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим знак
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить знак \"Стоп\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
```

```
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования знака
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- создадим знак "Стоп" на графике
   if(!ArrowStopCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
   int h_steps=bars*2/5;
//--- перемещаем точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d<bars-1)
         d+=1;
      //--- сдвигаем точку
      if(!ArrowStopMove(0,InpName,date[d],price[p]))
```

```
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.025 секунды
      Sleep(25);
     }
//--- меняем положение точки привязки относительно знака
   ArrowStopAnchorChange(0,InpName,ANCHOR_TOP);
//--- перерисуем график
   ChartRedraw();
//--- счетчик цикла
   h_steps=bars*2/5;
//--- перемещаем точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d<bars-1)
         d+=1;
      //--- сдвигаем точку
      if(!ArrowStopMove(0,InpName,date[d],price[p]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.025 секунды
      Sleep(25);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим знак с графика
   ArrowStopDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROW_STOP

Знак "Галка".

Примечание

Положение точки привязки относительно знака можно выбрать из перечисления ENUM_ARROW_ANCHOR.

Знаки большого размера (больше 5) можно создать только установив соответствующее значение свойства OBJPROP_WIDTH при написании кода в MetaEditor.

Пример

Следующий скрипт создает и перемещает на графике знак "Галка". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует знак \"Галка\"."
#property description "Координата точки привязки задается в"
#property description "процентах от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string             InpName="ArrowCheck"; // Имя знака
input int                InpDate=10;           // Дата точки привязки в %
input int                InpPrice=50;          // Цена точки привязки в %
input ENUM_ARROW_ANCHOR  InpAnchor=ANCHOR_TOP; // Способ привязки
input color              InpColor=clrRed;      // Цвет знака
```

```
input ENUM_LINE_STYLE   InpStyle=STYLE_DOT;   // Стиль окаймляющей линии
input int               InpWidth=5;           // Размер знака
input bool              InpBack=false;        // Знак на заднем плане
input bool              InpSelection=false;   // Выделить для перемещений
input bool              InpHidden=true;       // Скрыт в списке объектов
input long              InpZOrder=0;          // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает знак "Галка"                                             |
//+------------------------------------------------------------------+
bool ArrowCheckCreate(const long              chart_ID=0,           // ID графика
                      const string            name="ArrowCheck",    // имя знака
                      const int               sub_window=0,         // номер подокна
                      datetime                time=0,               // время точки пр
                      double                  price=0,              // цена точки при
                      const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // способ привязк
                      const color             clr=clrRed,           // цвет знака
                      const ENUM_LINE_STYLE   style=STYLE_SOLID,    // стиль окаймляю
                      const int               width=3,              // размер знака
                      const bool              back=false,           // на заднем план
                      const bool              selection=true,       // выделить для п
                      const bool              hidden=true,          // скрыт в списке
                      const long              z_order=0)            // приоритет на н
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим знак
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_CHECK,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать знак \"Галка\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет знака
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер знака
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowCheckMove(const long   chart_ID=0,        // ID графика
                    const string name="ArrowCheck", // имя объекта
                    datetime     time=0,            // координата времени точки привя
                    double       price=0)           // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Меняет способ привязки "Галки"                                   |
//+------------------------------------------------------------------+
bool ArrowCheckAnchorChange(const long              chart_ID=0,        // ID графика
                            const string            name="ArrowCheck", // имя объекта
                            const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ прив
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим способ привязки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
```

```
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет знак "Галка"                                             |
//+------------------------------------------------------------------+
bool ArrowCheckDelete(const long   chart_ID=0,        // ID графика
                      const string name="ArrowCheck") // имя знака
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим знак
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить знак \"Галка\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
```

```
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования знака
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- создадим знак "Галка" на графике
   if(!ArrowCheckCreate(0,InpName,0,date[d],price[p],InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки и менять ее положение относительно знака
//--- счетчик цикла
   int h_steps=bars*2/5;
//--- перемещаем точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d<bars-1)
         d+=1;
      //--- сдвигаем точку
      if(!ArrowCheckMove(0,InpName,date[d],price[p]))
```

```
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.025 секунды
      Sleep(25);
     }
//--- меняем положение точки привязки относительно знака
   ArrowCheckAnchorChange(0,InpName,ANCHOR_BOTTOM);
//--- перерисуем график
   ChartRedraw();
//--- счетчик цикла
   h_steps=bars*2/5;
//--- перемещаем точку привязки
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d<bars-1)
         d+=1;
      //--- сдвигаем точку
      if(!ArrowCheckMove(0,InpName,date[d],price[p]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.025 секунды
      Sleep(25);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим знак с графика
   ArrowCheckDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROW_LEFT_PRICE

Левая ценовая метка.

Пример

Следующий скрипт создает и перемещает на графике левую ценовую метку. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает левую ценовую метку на графике."
#property description "Координата точки привязки задается в процентах"
#property description "от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string            InpName="LeftPrice";  // Имя ценовой метки
input int               InpDate=100;          // Дата точки привязки в %
input int               InpPrice=10;          // Цена точки привязки в %
input color             InpColor=clrRed;      // Цвет ценовой метки
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID; // Стиль окаймляющей линии
input int               InpWidth=2;           // Размер ценовой метки
input bool              InpBack=false;        // Метка на заднем плане
input bool              InpSelection=true;    // Выделить для перемещений
input bool              InpHidden=true;       // Скрыт в списке объектов
input long              InpZOrder=0;          // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
```

```
//| Создает левую ценовую метку                                      |
//+------------------------------------------------------------------+
bool ArrowLeftPriceCreate(const long            chart_ID=0,        // ID графика
                          const string          name="LeftPrice",  // имя ценовой мет
                          const int             sub_window=0,      // номер подокна
                          datetime              time=0,            // время точки при
                          double                price=0,           // цена точки прив
                          const color           clr=clrRed,        // цвет ценовой ме
                          const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль окаймляющ
                          const int             width=1,           // размер ценовой
                          const bool            back=false,        // на заднем плане
                          const bool            selection=true,    // выделить для пе
                          const bool            hidden=true,       // скрыт в списке
                          const long            z_order=0)         // приоритет на на
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим ценовую метку
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_LEFT_PRICE,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать левую ценовую метку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет метки
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер метки
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
```

```
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowLeftPriceMove(const long   chart_ID=0,       // ID графика
                        const string name="LeftPrice", // имя метки
                        datetime     time=0,           // координата времени точки пр
                        double       price=0)          // координата цены точки привя
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет левую ценовую метку с графика                            |
//+------------------------------------------------------------------+
bool ArrowLeftPriceDelete(const long   chart_ID=0,       // ID графика
                          const string name="LeftPrice") // имя метки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим метку
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить левую ценовую метку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
```

```
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки метки
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования метки
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- создадим левую ценовую метку на графике
```

```
   if(!ArrowLeftPriceCreate(0,InpName,0,date[d],price[p],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки
//--- счетчик цикла
   int v_steps=accuracy*4/5;
//--- перемещаем точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p<accuracy-1)
         p+=1;
      //--- сдвигаем точку
      if(!ArrowLeftPriceMove(0,InpName,date[d],price[p]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим метку с графика
   ArrowLeftPriceDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROW_RIGHT_PRICE

Правая ценовая метка.

Пример

Следующий скрипт создает и перемещает на графике правую ценовую метку. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает правую ценовую метку на графике."
#property description "Координата точки привязки задается в процентах"
#property description "от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string            InpName="RightPrice"; // Имя ценовой метки
input int               InpDate=0;            // Дата точки привязки в %
input int               InpPrice=90;          // Цена точки привязки в %
input color             InpColor=clrRed;      // Цвет ценовой метки
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID; // Стиль окаймляющей линии
input int               InpWidth=2;           // Размер ценовой метки
input bool              InpBack=false;        // Метка на заднем плане
input bool              InpSelection=true;    // Выделить для перемещений
input bool              InpHidden=true;       // Скрыт в списке объектов
input long              InpZOrder=0;          // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
```

```
//| Создает правую ценовую метку                                     |
//+------------------------------------------------------------------+
bool ArrowRightPriceCreate(const long            chart_ID=0,        // ID графика
                           const string          name="RightPrice", // имя ценовой ме
                           const int             sub_window=0,      // номер подокна
                           datetime              time=0,            // время точки пр
                           double                price=0,           // цена точки при
                           const color           clr=clrRed,        // цвет ценовой м
                           const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль окаймляю
                           const int             width=1,           // размер ценовой
                           const bool            back=false,        // на заднем план
                           const bool            selection=true,    // выделить для п
                           const bool            hidden=true,       // скрыт в списке
                           const long            z_order=0)         // приоритет на н
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим ценовую метку
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_RIGHT_PRICE,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать правую ценовую метку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет метки
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер метки
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
```

```
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowRightPriceMove(const long   chart_ID=0,        // ID графика
                         const string name="RightPrice", // имя метки
                         datetime     time=0,            // координата времени точки
                         double       price=0)           // координата цены точки при
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет правую ценовую метку с графика                           |
//+------------------------------------------------------------------+
bool ArrowRightPriceDelete(const long   chart_ID=0,        // ID графика
                           const string name="RightPrice") // имя метки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим метку
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить правую ценовую метку! Код ошибки = ",GetLastError()
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
```

```
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки метки
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
      price[i]=min_price+i*step;
//--- определим точки для рисования метки
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- создадим правую ценовую метку на графике
```

```
   if(!ArrowRightPriceCreate(0,InpName,0,date[d],price[p],InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать точку привязки
//--- счетчик цикла
   int v_steps=accuracy*4/5;
//--- перемещаем точку привязки
   for(int i=0;i<v_steps;i++)
     {
      //--- возьмем следующее значение
      if(p>1)
         p-=1;
      //--- сдвигаем точку
      if(!ArrowRightPriceMove(0,InpName,date[d],price[p]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим метку с графика
   ArrowRightPriceDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_ARROW_BUY

Знак "Buy".

Пример

Следующий скрипт создает и перемещает на графике значки "Buy". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует значки \"Buy\" в окне графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input color InpColor=C'3,95,172'; // Цвет значков
//+------------------------------------------------------------------+
//| Создает знак "Buy"                                               |
//+------------------------------------------------------------------+
bool ArrowBuyCreate(const long            chart_ID=0,        // ID графика
                    const string          name="ArrowBuy",   // имя знака
                    const int             sub_window=0,      // номер подокна
                    datetime              time=0,            // время точки привязки
                    double                price=0,           // цена точки привязки
                    const color           clr=C'3,95,172',   // цвет знака
                    const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии (при выде
                    const int             width=1,           // размер линии (при выд
                    const bool            back=false,        // на заднем плане
```

```
                    const bool            selection=false,   // выделить для перемеще
                    const bool            hidden=true,       // скрыт в списке объект
                    const long            z_order=0)         // приоритет на нажатие
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим знак
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_BUY,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать знак \"Buy\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет знака
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии (при выделении)
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер линии (при выделении)
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowBuyMove(const long   chart_ID=0,       // ID графика
                  const string name="ArrowBuy", // имя объекта
                  datetime     time=0,          // координата времени точки привязки
                  double       price=0)         // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
```

```
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет знак "Buy"                                               |
//+------------------------------------------------------------------+
bool ArrowBuyDelete(const long   chart_ID=0,       // ID графика
                    const string name="ArrowBuy") // имя знака
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим знак
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить знак \"Buy\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // массив для хранения дат видимых баров
   double   low[];  // массив для хранения цен Low видимых баров
```

```
   double   high[]; // массив для хранения цен High видимых баров
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен Low
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Не удалось скопировать значения цен Low! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен High
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Не удалось скопировать значения цен High! Код ошибки = ",GetLastError())
      return;
     }
//--- создадим значки "Buy" в точке Low для каждого видимого бара
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyCreate(0,"ArrowBuy_"+(string)i,0,date[i],low[i],InpColor))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- переместим значки "Buy" в точку High для каждого видимого бара
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyMove(0,"ArrowBuy_"+(string)i,date[i],high[i]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
```

```
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- удалим значки "Buy"
   for(int i=0;i<bars;i++)
     {
      if(!ArrowBuyDelete(0,"ArrowBuy_"+(string)i))
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//---
  }
```

OBJ_ARROW_SELL

Знак "Sell".

Пример

Следующий скрипт создает и перемещает на графике значки "Sell". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт рисует значки \"Sell\" в окне графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input color InpColor=C'225,68,29'; // Цвет значков
//+------------------------------------------------------------------+
//| Создает знак "Sell"                                              |
//+------------------------------------------------------------------+
bool ArrowSellCreate(const long            chart_ID=0,        // ID графика
                     const string          name="ArrowSell",  // имя знака
                     const int             sub_window=0,      // номер подокна
                     datetime              time=0,            // время точки привязки
                     double                price=0,           // цена точки привязки
                     const color           clr=C'225,68,29',  // цвет знака
                     const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии (при выд
                     const int             width=1,           // размер линии (при вы
                     const bool            back=false,        // на заднем плане
```

```
                      const bool            selection=false,   // выделить для перемещ
                      const bool            hidden=true,       // скрыт в списке объек
                      const long            z_order=0)         // приоритет на нажатие
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим знак
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW_SELL,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать знак \"Sell\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим цвет знака
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии (при выделении)
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер линии (при выделении)
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения знака мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowSellMove(const long   chart_ID=0,        // ID графика
                   const string name="ArrowSell", // имя объекта
                   datetime     time=0,           // координата времени точки привязки
                   double       price=0)          // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
```

```
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет знак "Sell"                                              |
//+------------------------------------------------------------------+
bool ArrowSellDelete(const long   chart_ID=0,       // ID графика
                     const string name="ArrowSell") // имя знака
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим знак
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить знак \"Sell\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // массив для хранения дат видимых баров
   double   low[];  // массив для хранения цен Low видимых баров
```

```
   double   high[]; // массив для хранения цен High видимых баров
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен Low
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Не удалось скопировать значения цен Low! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен High
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Не удалось скопировать значения цен High! Код ошибки = ",GetLastError())
      return;
     }
//--- создадим значки "Sell" в точке High для каждого видимого бара
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellCreate(0,"ArrowSell_"+(string)i,0,date[i],high[i],InpColor))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- переместим значки "Sell" в точку Low для каждого видимого бара
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellMove(0,"ArrowSell_"+(string)i,date[i],low[i]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
```

```
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- удалим значки "Sell"
   for(int i=0;i<bars;i++)
     {
      if(!ArrowSellDelete(0,"ArrowSell_"+(string)i))
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//---
  }
```

OBJ_ARROW

Объект "Стрелка".

Примечание

Положение точки привязки относительно стрелки можно выбрать из перечисления ENUM_ARROW_ANCHOR.

Стрелки большого размера (больше 5) можно создать только установив соответствующее значение свойства OBJPROP_WIDTH при написании кода в MetaEditor.

Нужный тип стрелки можно выбрать, установив один из кодов символов шрифта Wingdings.

Пример

Следующий скрипт создает на графике объект "Стрелка" и изменяет ее тип. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает произвольную стрелку в окне графика."
#property description "Координата точки привязки задается в процентах"
#property description "от размеров окна графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string            InpName="Arrow";        // Имя стрелки
input int               InpDate=50;             // Дата точки привязки в %
input int               InpPrice=50;            // Цена точки привязки в %
```

```
input ENUM_ARROW_ANCHOR InpAnchor=ANCHOR_TOP;    // Способ привязки
input color             InpColor=clrDodgerBlue; // Цвет стрелки
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID;    // Стиль окаймляющей линии
input int               InpWidth=10;             // Размер стрелки
input bool              InpBack=false;           // Стрелка на заднем плане
input bool              InpSelection=false;      // Выделить для перемещений
input bool              InpHidden=true;          // Скрыт в списке объектов
input long              InpZOrder=0;             // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает стрелку                                                  |
//+------------------------------------------------------------------+
bool ArrowCreate(const long              chart_ID=0,           // ID графика
                 const string            name="Arrow",         // имя стрелки
                 const int               sub_window=0,         // номер подокна
                 datetime                time=0,               // время точки привязки
                 double                  price=0,              // цена точки привязки
                 const uchar             arrow_code=252,       // код стрелки
                 const ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM, // положение точки при
                 const color             clr=clrRed,           // цвет стрелки
                 const ENUM_LINE_STYLE   style=STYLE_SOLID,    // стиль окаймляющей л
                 const int               width=3,              // размер стрелки
                 const bool              back=false,           // на заднем плане
                 const bool              selection=true,       // выделить для переме
                 const bool              hidden=true,          // скрыт в списке объе
                 const long              z_order=0)            // приоритет на нажати
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeArrowEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим стрелку
   if(!ObjectCreate(chart_ID,name,OBJ_ARROW,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать стрелку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим код стрелки
   ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,arrow_code);
//--- установим способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет стрелки
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль окаймляющей линии
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер стрелки
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
```

```
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения стрелки мышью
//--- при создании графического объекта функцией ObjectCreate, по умолчанию объект
//--- нельзя выделить и перемещать. Внутри же этого метода параметр selection
//--- по умолчанию равен true, что позволяет выделять и перемещать этот объект
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool ArrowMove(const long   chart_ID=0,   // ID графика
               const string name="Arrow", // имя объекта
               datetime     time=0,       // координата времени точки привязки
               double       price=0)      // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Меняет код стрелки                                               |
//+------------------------------------------------------------------+
bool ArrowCodeChange(const long   chart_ID=0,   // ID графика
                     const string name="Arrow", // имя объекта
                     const uchar  code=252)     // од стрелки
  {
//--- сбросим значение ошибки
   ResetLastError();
```

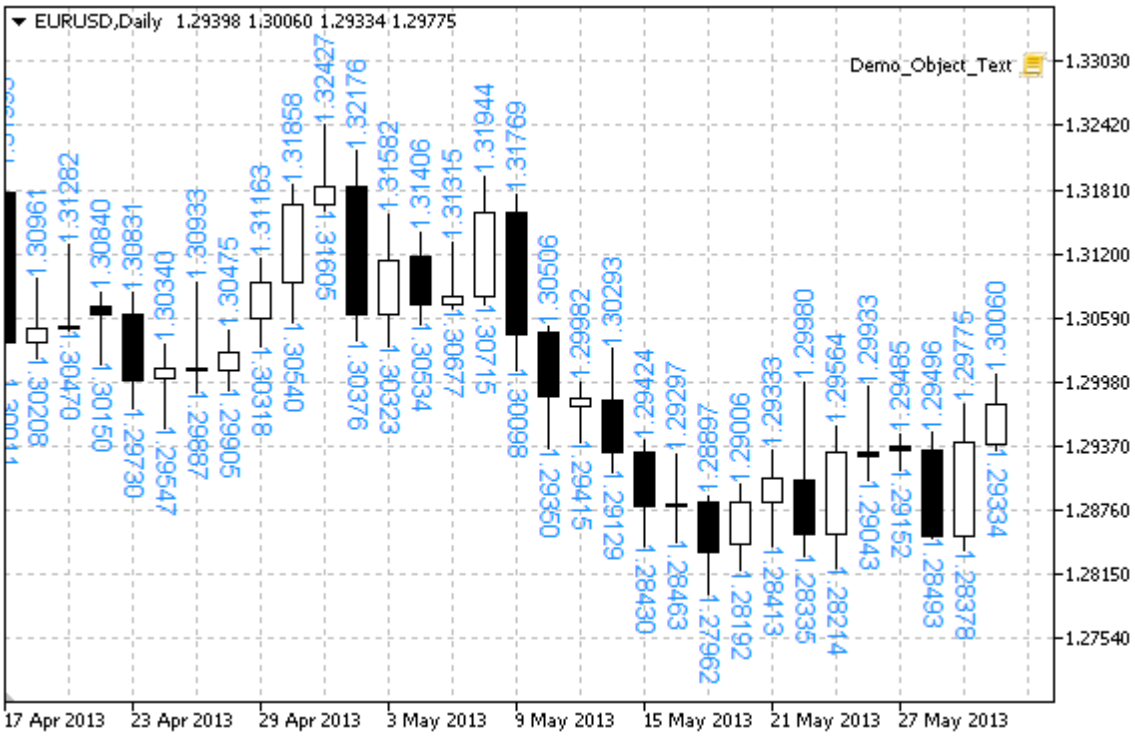
```
//--- изменим код стрелки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ARROWCODE,code))
     {
      Print(__FUNCTION__,
            ": не удалось изменить код стрелки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Меняет способ привязки                                           |
//+------------------------------------------------------------------+
bool ArrowAnchorChange(const long              chart_ID=0,        // ID графика
                       const string            name="Arrow",      // имя объекта
                       const ENUM_ARROW_ANCHOR anchor=ANCHOR_TOP) // способ привязки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим способ привязки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor))
     {
      Print(__FUNCTION__,
            ": не удалось изменить способ привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет стрелку                                                  |
//+------------------------------------------------------------------+
bool ArrowDelete(const long   chart_ID=0,   // ID графика
                 const string name="Arrow") // имя стрелки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим стрелку
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить стрелку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
```

```
//| Проверяет значения точки привязки, и для пустых значений          |
//| устанавливает значения по умолчанию                               |
//+------------------------------------------------------------------+
void ChangeArrowEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100 || InpPrice<0 || InpPrice>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- размер массива price
   int accuracy=1000;
//--- массивы для хранения значений дат и цен, которые будут использованы
//--- для установки и изменения координат точки привязки знака
   datetime date[];
   double   price[];
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(price,accuracy);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен
//--- найдем максимальное и минимальное значение графика
   double max_price=ChartGetDouble(0,CHART_PRICE_MAX);
   double min_price=ChartGetDouble(0,CHART_PRICE_MIN);
//--- определим шаг изменения цены и заполним массив
   double step=(max_price-min_price)/accuracy;
   for(int i=0;i<accuracy;i++)
```

```
      price[i]=min_price+i*step;
//--- определим точки для рисования стрелки
   int d=InpDate*(bars-1)/100;
   int p=InpPrice*(accuracy-1)/100;
//--- создадим стрелку на графике
   if(!ArrowCreate(0,InpName,0,date[d],price[p],32,InpAnchor,InpColor,
      InpStyle,InpWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график
   ChartRedraw();
//--- в цикле рассмотрим все варианты построения стрелок
   for(int i=33;i<256;i++)
     {
      if(!ArrowCodeChange(0,InpName,(uchar)i))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в пол секунды
      Sleep(500);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим стрелку с графика
   ArrowDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_TEXT

Объект "Текст".



Примечание

Положение точки привязки относительно текста можно выбрать из перечисления ENUM_ANCHOR_POINT. Также можно менять угол наклона текста при помощи свойства OBJPROP_ANGLE.

Пример

Следующий скрипт создает на графике несколько объектов "Текст". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает графический объект \"Текст\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string             InpFont="Arial";          // Шрифт
input int                InpFontSize=10;           // Размер шрифта
input color              InpColor=clrRed;          // Цвет
input double             InpAngle=90.0;            // Угол наклона в градусах
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_BOTTOM; // Способ привязки
input bool               InpBack=false;            // Объект на заднем плане
input bool               InpSelection=false;       // Выделить для перемещений
input bool               InpHidden=true;           // Скрыт в списке объектов
```

```
input long              InpZOrder=0;             // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает объект "Текст"                                           |
//+------------------------------------------------------------------+
bool TextCreate(const long              chart_ID=0,               // ID графика
                const string            name="Text",              // имя объекта
                const int               sub_window=0,             // номер подокна
                datetime                time=0,                   // время точки прив
                double                  price=0,                  // цена точки привя
                const string            text="Text",              // сам текст
                const string            font="Arial",             // шрифт
                const int               font_size=10,             // размер шрифта
                const color             clr=clrRed,               // цвет
                const double            angle=0.0,                // наклон текста
                const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // способ привязки
                const bool              back=false,               // на заднем плане
                const bool              selection=false,          // выделить для пер
                const bool              hidden=true,              // скрыт в списке о
                const long              z_order=0)                // приоритет на наж
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeTextEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим объект "Текст"
   if(!ObjectCreate(chart_ID,name,OBJ_TEXT,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать объект \"Текст\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим текст
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- установи шрифт текста
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- установим размер шрифта
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- установим угол наклона текста
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- установим способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения объекта мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
```

```
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает точку привязки                                        |
//+------------------------------------------------------------------+
bool TextMove(const long   chart_ID=0,  // ID графика
              const string name="Text", // имя объекта
              datetime     time=0,      // координата времени точки привязки
              double       price=0)     // координата цены точки привязки
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет текст объекта                                           |
//+------------------------------------------------------------------+
bool TextChange(const long   chart_ID=0,  // ID графика
                const string name="Text", // имя объекта
                const string text="Text") // текст
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим текст объекта
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": не удалось изменить текст! Код ошибки = ",GetLastError());
      return(false);
     }
```

```
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет объект "Текст"                                           |
//+------------------------------------------------------------------+
bool TextDelete(const long   chart_ID=0,  // ID графика
                const string name="Text") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим объект
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить объект \"Текст\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeTextEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[]; // массив для хранения дат видимых баров
   double   low[];  // массив для хранения цен Low видимых баров
   double   high[]; // массив для хранения цен High видимых баров
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(low,bars);
   ArrayResize(high,bars);
```

```
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен Low
   if(CopyLow(Symbol(),Period(),0,bars,low)==-1)
     {
      Print("Не удалось скопировать значения цен Low! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен High
   if(CopyHigh(Symbol(),Period(),0,bars,high)==-1)
     {
      Print("Не удалось скопировать значения цен High! Код ошибки = ",GetLastError())
      return;
     }
//--- определим как часто надо делать надписи
   int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- определим шаг
   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;
     }
//--- создадим надписи для значений High и Low баров (с промежутками)
   for(int i=0;i<bars;i+=step)
     {
      //--- создаем надписи
      if(!TextCreate(0,"TextHigh_"+(string)i,0,date[i],high[i],DoubleToString(high[i]
         InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
        {
         return;
        }
      if(!TextCreate(0,"TextLow_"+(string)i,0,date[i],low[i],DoubleToString(low[i],5)
```

```
          InpColor,-InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
         {
          return;
         }
       //--- проверим факт принудительного завершения скрипта
       if(IsStopped())
          return;
       //--- перерисуем график
       ChartRedraw();
       // задержка в 0.05 секунды
       Sleep(50);
      }
//--- задержка в пол секунды
   Sleep(500);
//--- удалим надписи
   for(int i=0;i<bars;i+=step)
     {
      if(!TextDelete(0,"TextHigh_"+(string)i))
         return;
      if(!TextDelete(0,"TextLow_"+(string)i))
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//---
  }
```

OBJ_LABEL

Объект "Текстовая метка".

Примечание

Положение точки привязки относительно метки можно выбрать из перечисления ENUM_ANCHOR_POINT. Координаты точки привязки задаются в пикселях.

Также можно выбрать угол привязки текстовой метки из перечисления ENUM_BASE_CORNER.

Пример

Следующий скрипт создает и перемещает на графике объект "Текстовая метка". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает графический объект \"Текстовая метка\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string            InpName="Label";          // Имя метки
input int               InpX=150;                 // Расстояние по оси X
input int               InpY=150;                 // Расстояние по оси Y
input string            InpFont="Arial";          // Шрифт
input int               InpFontSize=14;           // Размер шрифта
input color             InpColor=clrRed;          // Цвет
input double            InpAngle=0.0;             // Угол наклона в градусах
input ENUM_ANCHOR_POINT InpAnchor=ANCHOR_CENTER; // Способ привязки
```

```
input bool               InpBack=false;           // Объект на заднем плане
input bool               InpSelection=true;       // Выделить для перемещений
input bool               InpHidden=true;          // Скрыт в списке объектов
input long               InpZOrder=0;             // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает текстовую метку                                          |
//+------------------------------------------------------------------+
bool LabelCreate(const long              chart_ID=0,               // ID графика
                 const string            name="Label",             // имя метки
                 const int               sub_window=0,             // номер подокна
                 const int               x=0,                      // координата по о
                 const int               y=0,                      // координата по о
                 const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // угол графика дл
                 const string            text="Label",             // текст
                 const string            font="Arial",             // шрифт
                 const int               font_size=10,             // размер шрифта
                 const color             clr=clrRed,               // цвет
                 const double            angle=0.0,                // наклон текста
                 const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // способ привязки
                 const bool              back=false,               // на заднем плане
                 const bool              selection=false,          // выделить для пе
                 const bool              hidden=true,              // скрыт в списке
                 const long              z_order=0)                // приоритет на на
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим текстовую метку
   if(!ObjectCreate(chart_ID,name,OBJ_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать текстовую метку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим координаты метки
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим угол графика, относительно которого будут определяться координаты точ
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- установим текст
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- установи шрифт текста
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- установим размер шрифта
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- установим угол наклона текста
   ObjectSetDouble(chart_ID,name,OBJPROP_ANGLE,angle);
//--- установим способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
```

```
//--- установим цвет
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает текстовую метку                                       |
//+------------------------------------------------------------------+
bool LabelMove(const long   chart_ID=0,   // ID графика
               const string name="Label", // имя метки
               const int    x=0,          // координата по оси X
               const int    y=0)          // координата по оси Y
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим текстовую метку
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": не удалось переместить X координату метки! Код ошибки = ",GetLastError
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": не удалось переместить Y координату метки! Код ошибки = ",GetLastError
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет угол графика для привязки метки                         |
//+------------------------------------------------------------------+
bool LabelChangeCorner(const long             chart_ID=0,               // ID графика
                       const string           name="Label",             // имя метки
                       const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // угол графи
  {
//--- сбросим значение ошибки
```

```
   ResetLastError();
//--- изменим угол привязки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
     {
      Print(__FUNCTION__,
            ": не удалось изменить угол привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет текст объекта                                           |
//+------------------------------------------------------------------+
bool LabelTextChange(const long   chart_ID=0,   // ID графика
                     const string name="Label", // имя объекта
                     const string text="Text")  // текст
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим текст объекта
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": не удалось изменить текст! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет текстовую метку                                          |
//+------------------------------------------------------------------+
bool LabelDelete(const long   chart_ID=0,   // ID графика
                 const string name="Label") // имя метки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим метку
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить текстовую метку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
```

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- запомним координаты метки в локальные переменные
   int x=InpX;
   int y=InpY;
//--- размеры окна графика
   long x_distance;
   long y_distance;
//--- определим размеры окна
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
      return;
     }
//--- проверим входные параметры на корректность
   if(InpX<0 || InpX>x_distance-1 || InpY<0 || InpY>y_distance-1)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- подготовим начальный текст для метки
   string text;
   StringConcatenate(text,"Левый верхний угол: ",x,",",y);
//--- создадим текстовую метку на графике
   if(!LabelCreate(0,InpName,0,InpX,InpY,CORNER_LEFT_UPPER,text,InpFont,InpFontSize,
      InpColor,InpAngle,InpAnchor,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем пол секунды
   ChartRedraw();
   Sleep(500);
//--- будем перемещать метку и одновременно менять ее текст
//--- количество итераций по осям
   int h_steps=(int)(x_distance/2-InpX);
   int v_steps=(int)(y_distance/2-InpY);
//--- переместим метку вниз
   for(int i=0;i<v_steps;i++)
     {
      //--- меняем координату
```

```
      y+=2;
      //--- перемещаем метку и меняем ее текст
      MoveAndTextChange(x,y,"Левый верхний угол: ");
     }
//--- задержка в пол секунды
   Sleep(500);
//--- переместим метку вправо
   for(int i=0;i<h_steps;i++)
     {
      //--- меняем координату
      x+=2;
      //--- перемещаем метку и меняем ее текст
      MoveAndTextChange(x,y,"Левый верхний угол: ");
     }
//--- задержка в пол секунды
   Sleep(500);
//--- переместим метку вверх
   for(int i=0;i<v_steps;i++)
     {
      //--- меняем координату
      y-=2;
      //--- перемещаем метку и меняем ее текст
      MoveAndTextChange(x,y,"Левый верхний угол: ");
     }
//--- задержка в пол секунды
   Sleep(500);
//--- переместим метку влево
   for(int i=0;i<h_steps;i++)
     {
      //--- меняем координату
      x-=2;
      //--- перемещаем метку и меняем ее текст
      MoveAndTextChange(x,y,"Левый верхний угол: ");
     }
//--- задержка в пол секунды
   Sleep(500);
//--- теперь переместим точку путем изменения угла привязки
//--- переместим в левый нижний угол
   if(!LabelChangeCorner(0,InpName,CORNER_LEFT_LOWER))
      return;
//--- изменим текст метки
   StringConcatenate(text,"Левый нижний угол: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- перерисуем график и подождем две секунды
   ChartRedraw();
   Sleep(2000);
//--- переместим в правый нижний угол
```

```
   if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_LOWER))
      return;
//--- изменим текст метки
   StringConcatenate(text,"Правый нижний угол: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- перерисуем график и подождем две секунды
   ChartRedraw();
   Sleep(2000);
//--- переместим в правый верхний угол
   if(!LabelChangeCorner(0,InpName,CORNER_RIGHT_UPPER))
      return;
//--- изменим текст метки
   StringConcatenate(text,"Правый верхний угол: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- перерисуем график и подождем две секунды
   ChartRedraw();
   Sleep(2000);
//--- переместим в левый верхний угол
   if(!LabelChangeCorner(0,InpName,CORNER_LEFT_UPPER))
      return;
//--- изменим текст метки
   StringConcatenate(text,"Левый верхний угол: ",x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return;
//--- перерисуем график и подождем две секунды
   ChartRedraw();
   Sleep(2000);
//--- удалим метку
   LabelDelete(0,InpName);
//--- перерисуем график и подождем пол секунды
   ChartRedraw();
   Sleep(500);
//---
  }
//+------------------------------------------------------------------+
//| Функция перемещает объект и меняет его текст                     |
//+------------------------------------------------------------------+
bool MoveAndTextChange(const int x,const int y,string text)
  {
//--- перемещаем метку
   if(!LabelMove(0,InpName,x,y))
      return(false);
//--- изменим текст метки
   StringConcatenate(text,text,x,",",y);
   if(!LabelTextChange(0,InpName,text))
      return(false);
```

```
//--- проверим факт принудительного завершения скрипта
   if(IsStopped())
      return(false);
//--- перерисуем график
   ChartRedraw();
// задержка в 0.01 секунды
   Sleep(10);
//--- выход из функции
   return(true);
  }
```

OBJ_BUTTON

Объект "Кнопка".

Примечание

Координаты точки привязки задаются в пикселях. Можно выбрать угол привязки кнопки из перечисления ENUM_BASE_CORNER.

Пример

Следующий скрипт создает и перемещает на графике объект "Кнопка". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает кнопку на графике."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="Button";           // Имя кнопки
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Угол графика для привязки
input string           InpFont="Arial";            // Шрифт
input int              InpFontSize=14;             // Размер шрифта
input color            InpColor=clrBlack;          // Цвет текста
input color            InpBackColor=C'236,233,216'; // Цвет фона
input color            InpBorderColor=clrNONE;     // Цвет границы
input bool             InpState=false;             // Нажата/Отжата
input bool             InpBack=false;              // Объект на заднем плане
input bool             InpSelection=false;         // Выделить для перемещений
```

```
input bool              InpHidden=true;              // Скрыт в списке объектов
input long              InpZOrder=0;                 // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает кнопку                                                   |
//+------------------------------------------------------------------+
bool ButtonCreate(const long              chart_ID=0,               // ID графика
                  const string            name="Button",            // имя кнопки
                  const int               sub_window=0,             // номер подокна
                  const int               x=0,                      // координата по
                  const int               y=0,                      // координата по
                  const int               width=50,                 // ширина кнопки
                  const int               height=18,                // высота кнопки
                  const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // угол графика д
                  const string            text="Button",            // текст
                  const string            font="Arial",             // шрифт
                  const int               font_size=10,             // размер шрифта
                  const color             clr=clrBlack,             // цвет текста
                  const color             back_clr=C'236,233,216',  // цвет фона
                  const color             border_clr=clrNONE,       // цвет границы
                  const bool              state=false,              // нажата/отжата
                  const bool              back=false,               // на заднем плане
                  const bool              selection=false,          // выделить для п
                  const bool              hidden=true,              // скрыт в списке
                  const long              z_order=0)                // приоритет на н
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим кнопку
   if(!ObjectCreate(chart_ID,name,OBJ_BUTTON,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать кнопку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим координаты кнопки
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим размер кнопки
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим угол графика, относительно которого будут определяться координаты точ
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- установим текст
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- установи шрифт текста
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- установим размер шрифта
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
```

```
//--- установим цвет текста
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим цвет фона
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- установим цвет границы
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения кнопки мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает кнопку                                                |
//+------------------------------------------------------------------+
bool ButtonMove(const long   chart_ID=0,     // ID графика
                const string name="Button", // имя кнопки
                const int    x=0,           // координата по оси X
                const int    y=0)           // координата по оси Y
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим кнопку
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": не удалось переместить X координату кнопки! Код ошибки = ",GetLastErro
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": не удалось переместить Y координату кнопки! Код ошибки = ",GetLastErro
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет размер кнопки                                           |
//+------------------------------------------------------------------+
bool ButtonChangeSize(const long   chart_ID=0,     // ID графика
```

```
                      const string name="Button", // имя кнопки
                      const int    width=50,      // ширина кнопки
                      const int    height=18)     // высота кнопки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим размеры кнопки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": не удалось изменить ширину кнопки! Код ошибки = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": не удалось изменить высоту кнопки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет угол графика для привязки кнопки                        |
//+------------------------------------------------------------------+
bool ButtonChangeCorner(const long             chart_ID=0,               // ID графика
                        const string           name="Button",            // имя кнопки
                        const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER) // угол графи
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим угол привязки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner))
     {
      Print(__FUNCTION__,
            ": не удалось изменить угол привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет текст кнопки                                            |
//+------------------------------------------------------------------+
bool ButtonTextChange(const long   chart_ID=0,    // ID графика
                      const string name="Button", // имя кнопки
                      const string text="Text")   // текст
  {
```

```
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим текст объекта
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": не удалось изменить текст! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет кнопку                                                   |
//+------------------------------------------------------------------+
bool ButtonDelete(const long   chart_ID=0,    // ID графика
                  const string name="Button") // имя кнопки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим кнопку
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить кнопку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- размеры окна графика
   long x_distance;
   long y_distance;
//--- определим размеры окна
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
      return;
```

```
      }
//--- определим шаг для изменения размеров кнопки
   int x_step=(int)x_distance/32;
   int y_step=(int)y_distance/32;
//--- установим координаты кнопки и ее размер
   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;
//--- создадим кнопку
   if(!ButtonCreate(0,InpName,0,x,y,x_size,y_size,InpCorner,"Press",InpFont,InpFontSi
      InpColor,InpBackColor,InpBorderColor,InpState,InpBack,InpSelection,InpHidden,Inp
     {
      return;
     }
//--- перерисуем график
   ChartRedraw();
//--- в цикле уменьшаем кнопку
   int i=0;
   while(i<13)
     {
      //--- задержка в пол секунды
      Sleep(500);
      //--- переведем кнопку в нажатое состояние
      ObjectSetInteger(0,InpName,OBJPROP_STATE,true);
      //--- перерисуем график и подождем 0.2 секунды
      ChartRedraw();
      Sleep(200);
      //--- переопределим координаты и размер кнопки
      x+=x_step;
      y+=y_step;
      x_size-=x_step*2;
      y_size-=y_step*2;
      //--- уменьшим кнопку
      ButtonMove(0,InpName,x,y);
      ButtonChangeSize(0,InpName,x_size,y_size);
      //--- вернем кнопку в ненажатое состояние
      ObjectSetInteger(0,InpName,OBJPROP_STATE,false);
      //--- перерисуем график
      ChartRedraw();
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- увеличим счетчик цикла
      i++;
     }
//--- задержка в пол секунды
   Sleep(500);
```

```
//--- удалим кнопку
   ButtonDelete(0,InpName);
   ChartRedraw();
//--- подождем 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_CHART

Объект "График".

Примечание

Координаты точки привязки задаются в пикселях. Можно выбрать угол привязки из перечисления ENUM_BASE_CORNER.

Для объекта "График" можно выбирать символ, период и масштаб, а также включать/отключать режим отображения шкалы цены и даты.

Пример

Следующий скрипт создает и перемещает на графике объект "График". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает объект \"График\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="Chart";              // Имя объекта
input string           InpSymbol="EURUSD";           // Символ
input ENUM_TIMEFRAMES  InpPeriod=PERIOD_H1;          // Период
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Угол для привязки
input int              InpScale=2;                   // Масштаб
input bool             InpDateScale=true;            // Отображение шкалы времени
input bool             InpPriceScale=true;           // Отображение шкалы цены
```

```
input color            InpColor=clrRed;             // Цвет рамки при выделении
input ENUM_LINE_STYLE  InpStyle=STYLE_DASHDOTDOT;   // Стиль линии при выделении
input int              InpPointWidth=1;             // Размер точки для перемещений
input bool             InpBack=false;               // Объект на заднем плане
input bool             InpSelection=true;           // Выделить для перемещений
input bool             InpHidden=true;              // Скрыт в списке объектов
input long             InpZOrder=0;                 // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает объект "График"                                          |
//+------------------------------------------------------------------+
bool ObjectChartCreate(const long              chart_ID=0,               // ID графика
                       const string            name="Chart",             // имя объекта
                       const int               sub_window=0,             // номер подо
                       const string            symbol="EURUSD",          // символ
                       const ENUM_TIMEFRAMES   period=PERIOD_H1,         // период
                       const int               x=0,                      // координата
                       const int               y=0,                      // координата
                       const int               width=300,                // ширина
                       const int               height=200,               // высота
                       const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // угол для
                       const int               scale=2,                  // масштаб
                       const bool              date_scale=true,          // отображени
                       const bool              price_scale=true,         // отображени
                       const color             clr=clrRed,               // цвет рамки
                       const ENUM_LINE_STYLE   style=STYLE_SOLID,        // стиль лини
                       const int               point_width=1,            // размер точ
                       const bool              back=false,               // на заднем
                       const bool              selection=false,          // выделить
                       const bool              hidden=true,              // скрыт в с
                       const long              z_order=0)                // приоритет
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим объект "График"
   if(!ObjectCreate(chart_ID,name,OBJ_CHART,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать объект \"График\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим координаты объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим размер объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим угол графика, относительно которого будут определяться координаты точ
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
```

```
//--- установим символ
   ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol);
//--- установим период
   ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period);
//--- установим масштаб
   ObjectSetInteger(chart_ID,name,OBJPROP_CHART_SCALE,scale);
//--- отобразим (true) или скроем (false) шкалу времени
   ObjectSetInteger(chart_ID,name,OBJPROP_DATE_SCALE,date_scale);
//--- отобразим (true) или скроем (false) шкалу цены
   ObjectSetInteger(chart_ID,name,OBJPROP_PRICE_SCALE,price_scale);
//--- установим цвет рамки при включенном режиме выделения объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии рамки при включенном режиме выделения объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер точки привязки, с помощью которой можно перемещать объект
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Устанавливает символ и таймфрейм объекта "График"                |
//+------------------------------------------------------------------+
bool ObjectChartSetSymbolAndPeriod(const long            chart_ID=0,       // ID граф
                                   const string          name="Chart",     // имя объ
                                   const string          symbol="EURUSD",  // символ
                                   const ENUM_TIMEFRAMES period=PERIOD_H1) // таймфре
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- установим символ и таймфрейм объекта график
   if(!ObjectSetString(chart_ID,name,OBJPROP_SYMBOL,symbol))
     {
      Print(__FUNCTION__,
            ": не удалось установить символ для объекта \"График\"! Код ошибки = ",Ge
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_PERIOD,period))
     {
      Print(__FUNCTION__,
```

```
            ": не удалось установить период для объекта \"График\"! Код ошибки = ",Ge
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает объект "График"                                       |
//+------------------------------------------------------------------+
bool ObjectChartMove(const long   chart_ID=0,   // ID графика (не объекта)
                     const string name="Chart", // имя объекта
                     const int    x=0,          // координата по оси X
                     const int    y=0)          // координата по оси Y
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим объект
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": не удалось переместить X координату объекта \"График\"! Код ошибки = "
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": не удалось переместить Y координату объекта \"График\"! Код ошибки = "
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет размер объекта "График"                                 |
//+------------------------------------------------------------------+
bool ObjectChartChangeSize(const long   chart_ID=0,   // ID графика (не объекта)
                           const string name="Chart", // имя объекта
                           const int    width=300,    // ширина
                           const int    height=200)   // высота
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим размеры объекта
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": не удалось изменить ширину объекта \"График\"! Код ошибки = ",GetLastE
      return(false);
```

```
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": не удалось изменить высоту объекта \"График\"! Код ошибки = ",GetLastE
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Возвращает ID объекта "График"                                   |
//+------------------------------------------------------------------+
long ObjectChartGetID(const long   chart_ID=0,   // ID графика (не объекта)
                      const string name="Chart") // имя объекта
  {
//--- подготовим переменную для получения ID объекта "График"
   long id=-1;
//--- сбросим значение ошибки
   ResetLastError();
//--- получим ID
   if(!ObjectGetInteger(chart_ID,name,OBJPROP_CHART_ID,0,id))
     {
      Print(__FUNCTION__,
            ": не удалось получить ID объекта \"График\"! Код ошибки = ",GetLastError
     }
//--- возврат результата
   return(id);
  }
//+------------------------------------------------------------------+
//| Удаляет объект "График"                                          |
//+------------------------------------------------------------------+
bool ObjectChartDelete(const long   chart_ID=0,   // ID графика (не объекта)
                       const string name="Chart") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим кнопку
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить объект \"График\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
```

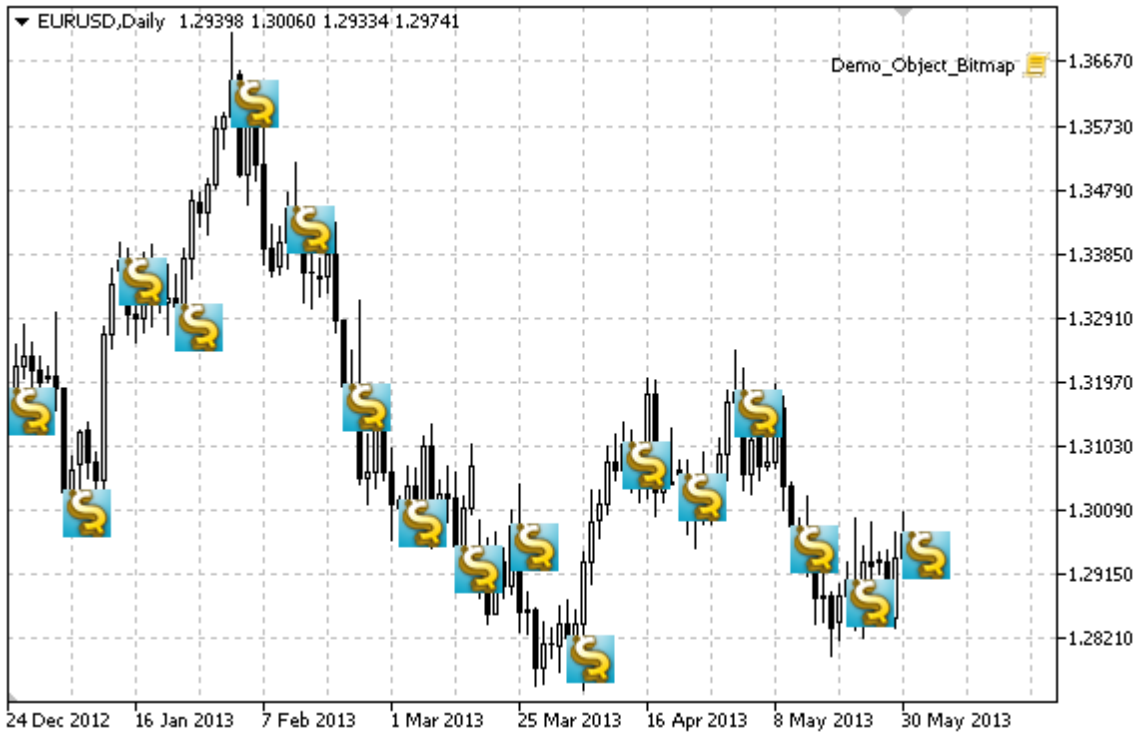
```
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- получим количество символов в "Обзоре рынка"
   int  symbols=SymbolsTotal(true);
//--- определим, есть ли символ с указанным именем в списке символов
   bool exist=false;
   for(int i=0;i<symbols;i++)
      if(InpSymbol==SymbolName(i,false))
        {
         exist=true;
         break;
        }
   if(!exist)
     {
      Print("Ошибка! Данный символ ",InpSymbol," не представлен в окне \"Обзор Рынка\
      return;
     }
//--- проверка входных параметров на корректность
   if(InpScale<0 || InpScale>5)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }

//--- размеры окна графика
   long x_distance;
   long y_distance;
//--- определим размеры окна
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
      return;
     }
//--- установим координаты объекта "График" и его размер
   int x=(int)x_distance/16;
   int y=(int)y_distance/16;
   int x_size=(int)x_distance*7/16;
   int y_size=(int)y_distance*7/16;
//--- создадим объект "График"
   if(!ObjectChartCreate(0,InpName,0,InpSymbol,InpPeriod,x,y,x_size,y_size,InpCorner,
      InpPriceScale,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,In
```

```
      {
       return;
      }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- растянем объект "График"
   int steps=(int)MathMin(x_distance*7/16,y_distance*7/16);
   for(int i=0;i<steps;i++)
     {
      //--- изменим размеры
      x_size+=1;
      y_size+=1;
      if(!ObjectChartChangeSize(0,InpName,x_size,y_size))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график и подождем 0.01 секунды
      ChartRedraw();
      Sleep(10);
     }
//--- задержка в пол секунды
   Sleep(500);
//--- изменим таймфрейм графика
   if(!ObjectChartSetSymbolAndPeriod(0,InpName,InpSymbol,PERIOD_M1))
      return;
   ChartRedraw();
//--- задержка в три секунды
   Sleep(3000);
//--- удалим объект
   ObjectChartDelete(0,InpName);
   ChartRedraw();
//--- подождем 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_BITMAP

Объект "Рисунок".



Примечание

Для объекта "Рисунок" можно выбирать область видимости рисунка.

Пример

Следующий скрипт создает на графике несколько картинок. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает рисунок в окне графика."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string          InpFile="\\Images\\dollar.bmp"; // Имя файла с картинкой
input int             InpWidth=24;                    // X координата области видимос
input int             InpHeight=24;                   // Y координата области видимос
input int             InpXOffset=4;                   // Смещение области видимости п
input int             InpYOffset=4;                   // Смещение области видимости п
input color           InpColor=clrRed;                // Цвет рамки при выделении
input ENUM_LINE_STYLE InpStyle=STYLE_SOLID;           // Стиль линии при выделении
input int             InpPointWidth=1;                // Размер точки для перемещений
input bool            InpBack=false;                  // Объект на заднем плане
input bool            InpSelection=false;             // Выделить для перемещений
```

```
input bool            InpHidden=true;                  // Скрыт в списке объектов
input long            InpZOrder=0;                     // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает рисунок в окне графика                                   |
//+------------------------------------------------------------------+
bool BitmapCreate(const long            chart_ID=0,        // ID графика
                  const string          name="Bitmap",     // имя рисунка
                  const int             sub_window=0,      // номер подокна
                  datetime              time=0,            // время точки привязки
                  double                price=0,           // цена точки привязки
                  const string          file="",           // имя файла с картинкой
                  const int             width=10,          // X координата области ви
                  const int             height=10,         // Y координата области ви
                  const int             x_offset=0,        // смещение области видимо
                  const int             y_offset=0,        // смещение области видимо
                  const color           clr=clrRed,        // цвет рамки при выделени
                  const ENUM_LINE_STYLE style=STYLE_SOLID, // стиль линии при выделен
                  const int             point_width=1,     // размер точки перемещени
                  const bool            back=false,        // на заднем плане
                  const bool            selection=false,   // выделить для перемещени
                  const bool            hidden=true,       // скрыт в списке объектов
                  const long            z_order=0)         // приоритет на нажатие мы
  {
//--- установим координаты точки привязки, если они не заданы
   ChangeBitmapEmptyPoint(time,price);
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим рисунок
   if(!ObjectCreate(chart_ID,name,OBJ_BITMAP,sub_window,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось создать рисунок в окне графика! Код ошибки = ",GetLastError
      return(false);
     }
//--- установим путь к файлу с картинкой
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
     {
      Print(__FUNCTION__,
            ": не удалось загрузить картинку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим область видимости изображения; если значения ширины или высоты
//--- больше значений ширины и высоты (соответственно) исходного изображения, то
//--- оно не рисуется; если значения ширины и высоты меньше размеров изображения,
//--- то рисуется та его часть, которая соответствует этим размерам
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим какая часть изображения должна показываться в области видимости
```

```
//--- по умолчанию это левая верхняя область изображения; значения позволяют
//--- произвести сдвиг от этого угла и отобразить другую часть изображения
   ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
   ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- установим цвет рамки при включенном режиме выделения объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии рамки при включенном режиме выделения объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер точки привязки, с помощью которой можно перемещать объект
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Устанавливает новую картинку для рисунка                         |
//+------------------------------------------------------------------+
bool BitmapSetImage(const long   chart_ID=0,     // ID графика
                    const string name="Bitmap", // имя рисунка
                    const string file="")       // путь к файлу
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- установим путь к файлу с картинкой
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,file))
     {
      Print(__FUNCTION__,
            ": не удалось загрузить картинку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает рисунок в окне графика                                |
//+------------------------------------------------------------------+
bool BitmapMove(const long   chart_ID=0,     // ID графика
                const string name="Bitmap", // имя рисунка
                datetime     time=0,        // время точки привязки
                double       price=0)       // цена точки привязки
```

```
  {
//--- если координаты точки не заданы, то перемещаем ее на текущий бар с ценой Bid
   if(!time)
      time=TimeCurrent();
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим точку привязки
   if(!ObjectMove(chart_ID,name,0,time,price))
     {
      Print(__FUNCTION__,
            ": не удалось переместить точку привязки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет размер области видимости (размер рисунка)               |
//+------------------------------------------------------------------+
bool BitmapChangeSize(const long   chart_ID=0,     // ID графика
                      const string name="Bitmap", // имя рисунка
                      const int    width=0,       // ширина рисунка
                      const int    height=0)      // высота рисунка
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим размеры рисунка
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": не удалось изменить ширину рисунка! Код ошибки = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": не удалось изменить высоту рисунка! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет координату левого верхнего угла области видимости       |
//+------------------------------------------------------------------+
bool BitmapMoveVisibleArea(const long   chart_ID=0,     // ID графика
```

```
                           const string name="Bitmap", // имя рисунка
                           const int    x_offset=0,    // координата X области видимо
                           const int    y_offset=0)    // координата Y области видимо
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим координаты области видимости рисунка
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
     {
      Print(__FUNCTION__,
            ": не удалось изменить X координату области видимости! Код ошибки = ",Get
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
     {
      Print(__FUNCTION__,
            ": не удалось изменить Y координату области видимости! Код ошибки = ",Get
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет рисунок                                                  |
//+------------------------------------------------------------------+
bool BitmapDelete(const long   chart_ID=0,     // ID графика
                  const string name="Bitmap") // имя рисунка
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим метку
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить рисунок! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Проверяет значения точки привязки, и для пустых значений         |
//| устанавливает значения по умолчанию                              |
//+------------------------------------------------------------------+
void ChangeBitmapEmptyPoint(datetime &time,double &price)
  {
//--- если время точки не задано, то она будет на текущем баре
   if(!time)
```

```
      time=TimeCurrent();
//--- если цена точки не задана, то она будет иметь значение Bid
   if(!price)
      price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date[];  // массив для хранения дат видимых баров
   double   close[]; // массив для хранения цен Close
//--- имя файла с картинкой
   string   file="\\Images\\dollar.bmp";
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- выделение памяти
   ArrayResize(date,bars);
   ArrayResize(close,bars);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- заполним массив цен Close
   if(CopyClose(Symbol(),Period(),0,bars,close)==-1)
     {
      Print("Не удалось скопировать значения цен Close! Код ошибки = ",GetLastError()
      return;
     }
//--- определим как часто надо выводить картинки
   int scale=(int)ChartGetInteger(0,CHART_SCALE);
//--- определим шаг
   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;
     }
//--- создадим рисунки для значений High и Low баров (с промежутками)
   for(int i=0;i<bars;i+=step)
     {
      //--- создаем рисунки
      if(!BitmapCreate(0,"Bitmap_"+(string)i,0,date[i],close[i],InpFile,InpWidth,InpH
         InpYOffset,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHidden,Inp
        {
         return;
        }
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в пол секунды
   Sleep(500);
//--- удалим значки "Sell"
   for(int i=0;i<bars;i+=step)
     {
      if(!BitmapDelete(0,"Bitmap_"+(string)i))
         return;
      if(!BitmapDelete(0,"Bitmap_"+(string)i))
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//---
  }
```

OBJ_BITMAP_LABEL

Объект "Графическая метка".

Примечание

Положение точки привязки относительно метки можно выбрать из перечисления ENUM_ANCHOR_POINT. Координаты точки привязки задаются в пикселях.

Также можно выбрать угол привязки графической метки из перечисления ENUM_BASE_CORNER.

Для объекта "Графическая метка" можно выбирать область видимости рисунка.

Пример

Следующий скрипт создает на графике несколько картинок. Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает объект \"Графическая метка\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string             InpName="BmpLabel";               // Имя метки
input string             InpFileOn="\\Images\\dollar.bmp"; // Имя файла для режима On
input string             InpFileOff="\\Images\\euro.bmp";  // Имя файла для режима Off
input bool               InpState=false;                   // Метка нажата/отжата
input ENUM_BASE_CORNER   InpCorner=CORNER_LEFT_UPPER;      // Угол графика для привязки
input ENUM_ANCHOR_POINT  InpAnchor=ANCHOR_CENTER;          // Способ привязки
input color              InpColor=clrRed;                  // Цвет рамки при выделении
```

```
input ENUM_LINE_STYLE   InpStyle=STYLE_SOLID;              // Стиль линии при выделени
input int               InpPointWidth=1;                   // Размер точки для перемещ
input bool              InpBack=false;                     // Объект на заднем плане
input bool              InpSelection=false;                // Выделить для перемещений
input bool              InpHidden=true;                    // Скрыт в списке объектов
input long              InpZOrder=0;                       // Приоритет на нажатие мыш
//+------------------------------------------------------------------+
//| Создает объект "Графическая метка"                               |
//+------------------------------------------------------------------+
bool BitmapLabelCreate(const long              chart_ID=0,               // ID график
                       const string            name="BmpLabel",          // имя метки
                       const int               sub_window=0,             // номер под
                       const int               x=0,                      // координат
                       const int               y=0,                      // координат
                       const string            file_on="",               // картинка
                       const string            file_off="",              // картинка
                       const int               width=0,                  // X координ
                       const int               height=0,                 // Y координ
                       const int               x_offset=10,              // смещение
                       const int               y_offset=10,              // смещение
                       const bool              state=false,              // нажата/от
                       const ENUM_BASE_CORNER  corner=CORNER_LEFT_UPPER, // угол граф
                       const ENUM_ANCHOR_POINT anchor=ANCHOR_LEFT_UPPER, // способ пр
                       const color             clr=clrRed,               // цвет рамк
                       const ENUM_LINE_STYLE   style=STYLE_SOLID,        // стиль лин
                       const int               point_width=1,            // размер то
                       const bool              back=false,               // на заднем
                       const bool              selection=false,          // выделить
                       const bool              hidden=true,              // скрыт в с
                       const long              z_order=0)                // приоритет
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим графическую иетку
   if(!ObjectCreate(chart_ID,name,OBJ_BITMAP_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать объект \"Графическая метка\"! Код ошибки = ",GetLas
      return(false);
     }
//--- установим картинки для режимов On и Off
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,0,file_on))
     {
      Print(__FUNCTION__,
            ": не удалось загрузить картинку для режима On! Код ошибки = ",GetLastErr
      return(false);
     }
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,1,file_off))
```

```
      {
       Print(__FUNCTION__,
             ": не удалось загрузить картинку для режима Off! Код ошибки = ",GetLastEr
       return(false);
      }
//--- установим координаты метки
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим область видимости изображения; если значения ширины или высоты
//--- больше значений ширины и высоты (соответственно) исходного изображения, то
//--- оно не рисуется; если значения ширины и высоты меньше размеров изображения,
//--- то рисуется та его часть, которая соответствует этим размерам
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим какая часть изображения должна показываться в области видимости
//--- по умолчанию это левая верхняя область изображения; значения позволяют
//--- произвести сдвиг от этого угла и отобразить другую часть изображения
   ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset);
   ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset);
//--- установим в каком сотояние находится метку (нажатое или отжатое)
   ObjectSetInteger(chart_ID,name,OBJPROP_STATE,state);
//--- установим угол графика, относительно которого будут определяться координаты точ
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- установим способ привязки
   ObjectSetInteger(chart_ID,name,OBJPROP_ANCHOR,anchor);
//--- установим цвет рамки при включенном режиме выделения объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим стиль линии рамки при включенном режиме выделения объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим размер точки привязки, с помощью которой можно перемещать объект
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,point_width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Устанавливает новую картинку для объекта "Графическая метка"     |
//+------------------------------------------------------------------+
bool BitmapLabelSetImage(const long   chart_ID=0,      // ID графика
                         const string name="BmpLabel", // имя метки
```

```
                          const int    on_off=0,         // модификатор (On или Off)
                          const string file="")          // путь к файлу
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- установим путь к файлу с картинкой
   if(!ObjectSetString(chart_ID,name,OBJPROP_BMPFILE,on_off,file))
     {
      Print(__FUNCTION__,
            ": не удалось загрузить картинку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает объект "Графическая метка"                            |
//+------------------------------------------------------------------+
bool BitmapLabelMove(const long   chart_ID=0,       // ID графика
                     const string name="BmpLabel", // имя метки
                     const int    x=0,             // координата по оси X
                     const int    y=0)             // координата по оси Y
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим объект
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": не удалось переместить X координату объекта! Код ошибки = ",GetLastErr
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": не удалось переместить Y координату объекта! Код ошибки = ",GetLastErr
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет размер области видимости (размер объекта)               |
//+------------------------------------------------------------------+
bool BitmapLabelChangeSize(const long   chart_ID=0,       // ID графика
                           const string name="BmpLabel", // имя метки
                           const int    width=0,         // ширина метки
                           const int    height=0)        // высота метки
```

```
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим размеры объекта
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": не удалось изменить ширину объекта! Код ошибки = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": не удалось изменить высоту объекта! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет координату левого верхнего угла области видимости       |
//+------------------------------------------------------------------+
bool BitmapLabelMoveVisibleArea(const long   chart_ID=0,       // ID графика
                                const string name="BmpLabel", // имя метки
                                const int    x_offset=0,      // координата X области
                                const int    y_offset=0)      // координата Y области
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим координаты области видимости объекта
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XOFFSET,x_offset))
     {
      Print(__FUNCTION__,
            ": не удалось изменить X координату области видимости! Код ошибки = ",Get
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YOFFSET,y_offset))
     {
      Print(__FUNCTION__,
            ": не удалось изменить Y координату области видимости! Код ошибки = ",Get
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет объект "Графическая метка"                               |
//+------------------------------------------------------------------+
```

```
bool BitmapLabelDelete(const long   chart_ID=0,      // ID графика
                       const string name="BmpLabel") // имя метки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим метку
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить объект \"Графическая метка\"! Код ошибки = ",GetLas
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- размеры окна графика
   long x_distance;
   long y_distance;
//--- определим размеры окна
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
      return;
     }
//--- определим координаты графической метки
   int x=(int)x_distance/2;
   int y=(int)y_distance/2;
//--- установим размеры метки и координаты области видимости
   int width=32;
   int height=32;
   int x_offset=0;
   int y_offset=0;
//--- разместим графическую метку по центру окна
   if(!BitmapLabelCreate(0,InpName,0,x,y,InpFileOn,InpFileOff,width,height,x_offset,y
      InpCorner,InpAnchor,InpColor,InpStyle,InpPointWidth,InpBack,InpSelection,InpHid
     {
      return;
     }
```

```
//--- перерисуем график и подождем одну секунду
   ChartRedraw();
   Sleep(1000);
//--- изменим размеры области видимости метки в цикле
   for(int i=0;i<6;i++)
     {
      //--- изменим размеры области видимости
      width--;
      height--;
      if(!BitmapLabelChangeSize(0,InpName,width,height))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.3 секунды
      Sleep(300);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- изменим координаты области видимости метки в цикле
   for(int i=0;i<2;i++)
     {
      //--- изменим координаты области видимости
      x_offset++;
      y_offset++;
      if(!BitmapLabelMoveVisibleArea(0,InpName,x_offset,y_offset))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.3 секунды
      Sleep(300);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим метку
   BitmapLabelDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_EDIT

Объект "Поле ввода".

Примечание

Координаты точки привязки задаются в пикселях. Можно выбрать угол привязки "Поля ввода" из перечисления ENUM_BASE_CORNER.

Также можно выбрать один из типов выравнивания текста внутри "Поля ввода" из перечисления ENUM_ALIGN_MODE.

Пример

Следующий скрипт создает и перемещает на графике объект "Поле ввода". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает объект \"Поле ввода\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string            InpName="Edit";              // Имя объекта
input string            InpText="Text";              // Текст объекта
input string            InpFont="Arial";             // Шрифт
input int               InpFontSize=14;              // Размер шрифта
input ENUM_ALIGN_MODE   InpAlign=ALIGN_CENTER;       // Способ выравнивания текста
input bool              InpReadOnly=false;           // Возможность редактировать
input ENUM_BASE_CORNER  InpCorner=CORNER_LEFT_UPPER; // Угол графика для привязки
input color             InpColor=clrBlack;           // Цвет текста
```

```
input color            InpBackColor=clrWhite;       // Цвет фона
input color            InpBorderColor=clrBlack;     // Цвет границы
input bool             InpBack=false;               // Объект на заднем плане
input bool             InpSelection=false;          // Выделить для перемещений
input bool             InpHidden=true;              // Скрыт в списке объектов
input long             InpZOrder=0;                 // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает объект "Поле ввода"                                      |
//+------------------------------------------------------------------+
bool EditCreate(const long             chart_ID=0,               // ID графика
                const string           name="Edit",              // имя объекта
                const int              sub_window=0,             // номер подокна
                const int              x=0,                      // координата по оси
                const int              y=0,                      // координата по оси
                const int              width=50,                 // ширина
                const int              height=18,                // высота
                const string           text="Text",              // текст
                const string           font="Arial",             // шрифт
                const int              font_size=10,             // размер шрифта
                const ENUM_ALIGN_MODE  align=ALIGN_CENTER,       // способ выравнивани
                const bool             read_only=false,          // возможность редак
                const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // угол графика для
                const color            clr=clrBlack,             // цвет текста
                const color            back_clr=clrWhite,        // цвет фона
                const color            border_clr=clrNONE,       // цвет границы
                const bool             back=false,               // на заднем плане
                const bool             selection=false,          // выделить для пере
                const bool             hidden=true,              // скрыт в списке об
                const long             z_order=0)                // приоритет на нажа
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим поле ввода
   if(!ObjectCreate(chart_ID,name,OBJ_EDIT,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать объект \"Поле ввода\"! Код ошибки = ",GetLastError(
      return(false);
     }
//--- установим координаты объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим размеры объекта
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим текст
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- установи шрифт текста
```

```
   ObjectSetString(chart_ID,name,OBJPROP_FONT,font);
//--- установим размер шрифта
   ObjectSetInteger(chart_ID,name,OBJPROP_FONTSIZE,font_size);
//--- установим способ выравнивания текста в объекте
   ObjectSetInteger(chart_ID,name,OBJPROP_ALIGN,align);
//--- установим (true) или отменим (false) режим только для чтения
   ObjectSetInteger(chart_ID,name,OBJPROP_READONLY,read_only);
//--- установим угол графика, относительно которого будут определяться координаты объ
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- установим цвет текста
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим цвет фона
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- установим цвет границы
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_COLOR,border_clr);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает объект "Поле ввода"                                   |
//+------------------------------------------------------------------+
bool EditMove(const long   chart_ID=0,  // ID графика
              const string name="Edit", // имя объекта
              const int    x=0,         // координата по оси X
              const int    y=0)         // координата по оси Y
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим объект
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": не удалось переместить X координату объекта! Код ошибки = ",GetLastErr
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": не удалось переместить Y координату объекта! Код ошибки = ",GetLastErr
```

```
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет размеры объекта "Поле ввода"                            |
//+------------------------------------------------------------------+
bool EditChangeSize(const long   chart_ID=0,  // ID графика
                    const string name="Edit", // имя объекта
                    const int    width=0,     // ширина
                    const int    height=0)    // высота
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим размеры объекта
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": не удалось изменить ширину объекта! Код ошибки = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": не удалось изменить высоту объекта! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет текст объекта "Поле ввода"                              |
//+------------------------------------------------------------------+
bool EditTextChange(const long   chart_ID=0,  // ID графика
                    const string name="Edit", // имя объекта
                    const string text="Text") // текст
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим текст объекта
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": не удалось изменить текст! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Возвращает текст объекта "Поле ввода"                            |
//+------------------------------------------------------------------+
bool EditTextGet(string      &text,        // текст
                 const long   chart_ID=0,  // ID графика
                 const string name="Edit") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- получим текст объекта
   if(!ObjectGetString(chart_ID,name,OBJPROP_TEXT,0,text))
     {
      Print(__FUNCTION__,
            ": не удалось получить текст! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет объект "Поле ввода"                                      |
//+------------------------------------------------------------------+
bool EditDelete(const long   chart_ID=0,  // ID графика
                const string name="Edit") // имя объекта
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим метку
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить объект \"Поле ввода\"! Код ошибки = ",GetLastError(
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- размеры окна графика
   long x_distance;
   long y_distance;
//--- определим размеры окна
```

```
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
      return;
     }
//--- определим шаг для изменения размеров поля ввода
   int x_step=(int)x_distance/64;
//--- установим координаты поля ввода и его размер
   int x=(int)x_distance/8;
   int y=(int)y_distance/2;
   int x_size=(int)x_distance/8;
   int y_size=InpFontSize*2;
//--- запомним текст в локальную переменную
   string text=InpText;
//--- создадим поле ввода
   if(!EditCreate(0,InpName,0,x,y,x_size,y_size,InpText,InpFont,InpFontSize,InpAlign,
      InpCorner,InpColor,InpBackColor,InpBorderColor,InpBack,InpSelection,InpHidden,I
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- растянем поле ввода
   while(x_size-x<x_distance*5/8)
     {
      //--- увеличим ширину поля ввода
      x_size+=x_step;
      if(!EditChangeSize(0,InpName,x_size,y_size))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график и подождем 0.05 секунды
      ChartRedraw();
      Sleep(50);
     }
//--- задержка в пол секунды
   Sleep(500);
//--- изменим текст
   for(int i=0;i<20;i++)
     {
      //--- прибавим "+" в начале и конце
```

```
      text="+"+text+"+";
      if(!EditTextChange(0,InpName,text))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график и подождем 0.1 секунды
      ChartRedraw();
      Sleep(100);
     }
//--- задержка в пол секунды
   Sleep(500);
//--- удалим поле ввода
   EditDelete(0,InpName);
   ChartRedraw();
//--- подождем 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_EVENT

Объект "Событие".

Примечание

При наведении на событие мышью, всплывает его текст.

Пример

Следующий скрипт создает и перемещает на графике объект "Событие". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт строит графический объект \"Событие\"."
#property description "Дата точки привязки задается в процентах от"
#property description "ширины окна графика в барах."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="Event";     // Имя события
input int              InpDate=25;          // Дата события в %
input string           InpText="Text";      // Текст события
input color            InpColor=clrRed;     // Цвет события
input int              InpWidth=1;          // Размер точки при выделении
input bool             InpBack=false;       // Событие на заднем плане
input bool             InpSelection=false; // Выделить для перемещений
input bool             InpHidden=true;      // Скрыт в списке объектов
```

```
input long            InpZOrder=0;        // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает объект "Событие" на графике                              |
//+------------------------------------------------------------------+
bool EventCreate(const long            chart_ID=0,       // ID графика
                 const string          name="Event",     // имя события
                 const int             sub_window=0,     // номер подокна
                 const string          text="Text",      // текст события
                 datetime              time=0,           // время
                 const color           clr=clrRed,       // цвет
                 const int             width=1,          // толщина точки при выделени
                 const bool            back=false,       // на заднем плане
                 const bool            selection=false,  // выделить для перемещений
                 const bool            hidden=true,      // скрыт в списке объектов
                 const long            z_order=0)        // приоритет на нажатие мышью
  {
//--- если время не задано, то создаем объект на последнем баре
   if(!time)
      time=TimeCurrent();
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим объект "Событие"
   if(!ObjectCreate(chart_ID,name,OBJ_EVENT,sub_window,time,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать объект \"Событие\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим текст события
   ObjectSetString(chart_ID,name,OBJPROP_TEXT,text);
//--- установим цвет
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
//--- установим толщину точки привязки если объект выделен
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения события мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет текст объекта "Событие"                                 |
```

```
//+------------------------------------------------------------------+
bool EventTextChange(const long   chart_ID=0,   // ID графика
                     const string name="Event", // имя события
                     const string text="Text")  // текст
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим текст объекта
   if(!ObjectSetString(chart_ID,name,OBJPROP_TEXT,text))
     {
      Print(__FUNCTION__,
            ": не удалось изменить текст! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещение объект "Событие"                                     |
//+------------------------------------------------------------------+
bool EventMove(const long   chart_ID=0,   // ID графика
               const string name="Event", // имя события
               datetime     time=0)       // время
  {
//--- если время не задано, то перемещаем событие на последний бар
   if(!time)
      time=TimeCurrent();
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим объект
   if(!ObjectMove(chart_ID,name,0,time,0))
     {
      Print(__FUNCTION__,
            ": не удалось переместить объект \"Событие\"! Код ошибки = ",GetLastError
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет объект "Событие"                                         |
//+------------------------------------------------------------------+
bool EventDelete(const long   chart_ID=0,   // ID графика
                 const string name="Event") // имя события
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим объект
```

```
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить объект \"Событие\"! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- проверим входные параметры на корректность
   if(InpDate<0 || InpDate>100)
     {
      Print("Ошибка! Некорректные значения входных параметров!");
      return;
     }
//--- количество видимых баров в окне графика
   int bars=(int)ChartGetInteger(0,CHART_VISIBLE_BARS);
//--- массив для хранения значений дат, которые будут использованы
//--- для установки и изменения координаты точки привязки линии
   datetime date[];
//--- выделение памяти
   ArrayResize(date,bars);
//--- заполним массив дат
   ResetLastError();
   if(CopyTime(Symbol(),Period(),0,bars,date)==-1)
     {
      Print("Не удалось скопировать значения времени! Код ошибки = ",GetLastError());
      return;
     }
//--- определим точки для создания объекта
   int d=InpDate*(bars-1)/100;
//--- создадим объект "Событие"
   if(!EventCreate(0,InpName,0,InpText,date[d],InpColor,InpWidth,
      InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- теперь будем перемещать объект
//--- счетчик цикла
   int h_steps=bars/2;
```

```
//--- перемещаем объект
   for(int i=0;i<h_steps;i++)
     {
      //--- возьмем следующее значение
      if(d<bars-1)
         d+=1;
      //--- сдвигаем точку
      if(!EventMove(0,InpName,date[d]))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график
      ChartRedraw();
      // задержка в 0.05 секунды
      Sleep(50);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- удалим канал с графика
   EventDelete(0,InpName);
   ChartRedraw();
//--- задержка в 1 секунду
   Sleep(1000);
//---
  }
```

OBJ_RECTANGLE_LABEL

Объект "Прямоугольная метка".

Примечание

Координаты точки привязки задаются в пикселях. Можно выбрать угол привязки прямоугольной метки из перечисления ENUM_BASE_CORNER. Тип рамки для прямоугольной метки можно выбрать из перечисления ENUM_BORDER_TYPE.

Предназначена для создания и оформления пользовательского графического интерфейса.

Пример

Следующий скрипт создает и перемещает на графике объект "Прямоугольная метка". Для создания и изменения свойств графического объекта написаны специальные функции, которые вы можете использовать "как есть" в своих собственных программах.

```
//--- описание
#property description "Скрипт создает графический объект \"Прямоугольная метка\"."
//--- покажем окно входных параметров при запуске скрипта
#property script_show_inputs
//--- входные параметры скрипта
input string           InpName="RectLabel";          // Имя метки
input color            InpBackColor=clrSkyBlue;      // Цвет фона
input ENUM_BORDER_TYPE InpBorder=BORDER_FLAT;        // Тип рамки
input ENUM_BASE_CORNER InpCorner=CORNER_LEFT_UPPER; // Угол графика для привязки
input color            InpColor=clrDarkBlue;         // Цвет плоской границы (Flat)
input ENUM_LINE_STYLE  InpStyle=STYLE_SOLID;         // Стиль плоской границы (Flat)
input int              InpLineWidth=3;               // Толщина плоской границы (Flat)
```

```
input bool              InpBack=false;              // Объект на заднем плане
input bool              InpSelection=true;          // Выделить для перемещений
input bool              InpHidden=true;             // Скрыт в списке объектов
input long              InpZOrder=0;                // Приоритет на нажатие мышью
//+------------------------------------------------------------------+
//| Создает прямоугольную метку                                      |
//+------------------------------------------------------------------+
bool RectLabelCreate(const long             chart_ID=0,               // ID графика
                     const string           name="RectLabel",         // имя метки
                     const int              sub_window=0,             // номер подокна
                     const int              x=0,                      // координата по
                     const int              y=0,                      // координата по
                     const int              width=50,                 // ширина
                     const int              height=18,                // высота
                     const color            back_clr=C'236,233,216',  // цвет фона
                     const ENUM_BORDER_TYPE border=BORDER_SUNKEN,     // тип границы
                     const ENUM_BASE_CORNER corner=CORNER_LEFT_UPPER, // угол графика
                     const color            clr=clrRed,               // цвет плоской
                     const ENUM_LINE_STYLE  style=STYLE_SOLID,        // стиль плоско
                     const int              line_width=1,             // толщина плос
                     const bool             back=false,               // на заднем пл
                     const bool             selection=false,          // выделить для
                     const bool             hidden=true,              // скрыт в спис
                     const long             z_order=0)                // приоритет на
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- создадим прямоугольную метку
   if(!ObjectCreate(chart_ID,name,OBJ_RECTANGLE_LABEL,sub_window,0,0))
     {
      Print(__FUNCTION__,
            ": не удалось создать прямоугольную метку! Код ошибки = ",GetLastError());
      return(false);
     }
//--- установим координаты метки
   ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x);
   ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y);
//--- установим размеры метки
   ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width);
   ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height);
//--- установим цвет фона
   ObjectSetInteger(chart_ID,name,OBJPROP_BGCOLOR,back_clr);
//--- установим тип границы
   ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border);
//--- установим угол графика, относительно которого будут определяться координаты точ
   ObjectSetInteger(chart_ID,name,OBJPROP_CORNER,corner);
//--- установим цвет плоской рамки (в режиме Flat)
   ObjectSetInteger(chart_ID,name,OBJPROP_COLOR,clr);
```

```
//--- установим стиль линии плоской рамки
   ObjectSetInteger(chart_ID,name,OBJPROP_STYLE,style);
//--- установим толщину плоской границы
   ObjectSetInteger(chart_ID,name,OBJPROP_WIDTH,line_width);
//--- отобразим на переднем (false) или заднем (true) плане
   ObjectSetInteger(chart_ID,name,OBJPROP_BACK,back);
//--- включим (true) или отключим (false) режим перемещения метки мышью
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTABLE,selection);
   ObjectSetInteger(chart_ID,name,OBJPROP_SELECTED,selection);
//--- скроем (true) или отобразим (false) имя графического объекта в списке объектов
   ObjectSetInteger(chart_ID,name,OBJPROP_HIDDEN,hidden);
//--- установи приоритет на получение события нажатия мыши на графике
   ObjectSetInteger(chart_ID,name,OBJPROP_ZORDER,z_order);
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Перемещает прямоугольную метку                                   |
//+------------------------------------------------------------------+
bool RectLabelMove(const long   chart_ID=0,        // ID графика
                   const string name="RectLabel", // имя метки
                   const int    x=0,              // координата по оси X
                   const int    y=0)              // координата по оси Y
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- переместим прямоугольную метку
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XDISTANCE,x))
     {
      Print(__FUNCTION__,
            ": не удалось переместить X координату метки! Код ошибки = ",GetLastError
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YDISTANCE,y))
     {
      Print(__FUNCTION__,
            ": не удалось переместить Y координату метки! Код ошибки = ",GetLastError
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет размер прямоугольной метки                              |
//+------------------------------------------------------------------+
bool RectLabelChangeSize(const long   chart_ID=0,        // ID графика
                         const string name="RectLabel", // имя метки
                         const int    width=50,         // ширина метки
```

```
                          const int     height=18)         // высота метки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим размеры метки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_XSIZE,width))
     {
      Print(__FUNCTION__,
            ": не удалось изменить ширину метки! Код ошибки = ",GetLastError());
      return(false);
     }
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_YSIZE,height))
     {
      Print(__FUNCTION__,
            ": не удалось изменить высоту метки! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Изменяет тип границы прямоугольной метки                         |
//+------------------------------------------------------------------+
bool RectLabelChangeBorderType(const long             chart_ID=0,            // ID гра
                               const string           name="RectLabel",      // имя ме
                               const ENUM_BORDER_TYPE border=BORDER_SUNKEN) // тип гр
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- изменим тип рамки
   if(!ObjectSetInteger(chart_ID,name,OBJPROP_BORDER_TYPE,border))
     {
      Print(__FUNCTION__,
            ": не удалось изменить тип границы! Код ошибки = ",GetLastError());
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Удаляет прямоугольную метку                                      |
//+------------------------------------------------------------------+
bool RectLabelDelete(const long   chart_ID=0,       // ID графика
                     const string name="RectLabel") // имя метки
  {
//--- сбросим значение ошибки
   ResetLastError();
//--- удалим метку
```

```
   if(!ObjectDelete(chart_ID,name))
     {
      Print(__FUNCTION__,
            ": не удалось удалить прямоугольную метку! Код ошибки = ",GetLastError())
      return(false);
     }
//--- успешное выполнение
   return(true);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- размеры окна графика
   long x_distance;
   long y_distance;
//--- определим размеры окна
   if(!ChartGetInteger(0,CHART_WIDTH_IN_PIXELS,0,x_distance))
     {
      Print("Не удалось получить ширину графика! Код ошибки = ",GetLastError());
      return;
     }
   if(!ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0,y_distance))
     {
      Print("Не удалось получить высоту графика! Код ошибки = ",GetLastError());
      return;
     }
//--- определим координаты прямоугольной метки
   int x=(int)x_distance/4;
   int y=(int)y_distance/4;
//--- установим размеры метки
   int width=(int)x_distance/4;
   int height=(int)y_distance/4;
//--- создадим прямоугольную метку
   if(!RectLabelCreate(0,InpName,0,x,y,width,height,InpBackColor,InpBorder,InpCorner,
      InpColor,InpStyle,InpLineWidth,InpBack,InpSelection,InpHidden,InpZOrder))
     {
      return;
     }
//--- перерисуем график и подождем одну секунду
   ChartRedraw();
   Sleep(1000);
//--- изменим размер прямоугольной метки
   int steps=(int)MathMin(x_distance/4,y_distance/4);
   for(int i=0;i<steps;i++)
     {
      //--- изменим размеры
```

```
      width+=1;
      height+=1;
      if(!RectLabelChangeSize(0,InpName,width,height))
         return;
      //--- проверим факт принудительного завершения скрипта
      if(IsStopped())
         return;
      //--- перерисуем график и подождем 0.01 секунды
      ChartRedraw();
      Sleep(10);
     }
//--- задержка в 1 секунду
   Sleep(1000);
//--- изменим тип рамки
   if(!RectLabelChangeBorderType(0,InpName,BORDER_RAISED))
      return;
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- изменим тип рамки
   if(!RectLabelChangeBorderType(0,InpName,BORDER_SUNKEN))
      return;
//--- перерисуем график и подождем 1 секунду
   ChartRedraw();
   Sleep(1000);
//--- удалим метку
   RectLabelDelete(0,InpName);
   ChartRedraw();
//--- подождем 1 секунду
   Sleep(1000);
//---
  }
```

Propiedades de objetos

Cada objeto gráfico en el gráfico de precios tiene un conjunto de propiedades. La definición y obtención de los valores de las propiedades de objetos se realiza a través de las funciones para trabajar con objetos gráficos. Para cada tipo de objeto existe su propio conjunto de propiedades. Más abajo viene la lista de todos los posibles valores de la familia de enumeraciones ENUM_OBJECT_PROPERTY. Algunas propiedades requieren la aclaración, como por ejemplo, el número de nivel para el objeto de la extensión Fibonacci. En estos casos, es necesario especificar el valor del parámetro *modifier* en las funciones ObjectSet...() y ObjectGet...().

Para las funciones ObjectSetInteger() y ObjectGetInteger()

ENUM_OBJECT_PROPERTY_INTEGER

Identificador	Descripción	Tipo de la propiedad
OBJPROP_COLOR	Color	color
OBJPROP_STYLE	Estilo	ENUM_LINE_STYLE
OBJPROP_WIDTH	Grosor de línea	int
OBJPROP_BACK	Objeto en el fondo	bool
OBJPROP_ZORDER	La prioridad de un objeto gráfico para obtener el evento de cliquear sobre el gráfico (CHARTEVENT_CLICK). Por defecto, cuando se crea un objeto, este valor se pone a cero; pero si hace falta, se puede subir la prioridad. Cuando los objetos se aplican uno al otro, sólo uno de ellos, cuya prioridad es superior, recibirá el evento CHARTEVENT_CLICK.	long
OBJPROP_FILL	Relleno de objeto con color (para OBJ_RECTANGLE, OBJ_TRIANGLE, OBJ_ELLIPSE, OBJ_CHANNEL, OBJ_STDDEVCHANNEL, OBJ_REGRESSION)	bool
OBJPROP_HIDDEN	Prohíbe mostrar el nombre del objeto gráfico en la lista de objetos del menú del terminal "Gráficos" - "Objetos" - "Lista de objetos". El valor true permite ocultar el objeto que el usuario no necesita. Por defecto, true se pone para los objetos que muestran los	bool

	eventos del calendario, historial de trading, así como para los objetos creados en un programa MQL5. Para poder ver estos objetos gráfico y acceder a sus propiedades, hay que pulsar en el botón "Todos" en la ventana "Lista de objetos".	
OBJPROP_SELECTED	Selección de objeto	bool
OBJPROP_READONLY	Posibilidad de editar el texto en el objeto Edit	bool
OBJPROP_TYPE	Tipo de objeto	ENUM_OBJECT r/o
OBJPROP_TIME	Coordenadas de tiempo	datetime modificador=número del punto de anclaje
OBJPROP_SELECTABLE	Disponibilidad de objeto	bool
OBJPROP_CREATETIME	Tiempo de creación de objeto	datetime r/o
OBJPROP_LEVELS	Número de niveles	int
OBJPROP_LEVELCOLOR	Color de línea-nivel	color modificador=número del nivel
OBJPROP_LEVELSTYLE	Estilo de línea-nivel	ENUM_LINE_STYLE modificador=número del nivel
OBJPROP_LEVELWIDTH	Grosor de línea-nivel	int modificador=número del nivel
OBJPROP_ALIGN	Alineación del texto horizontalmente en el objeto "Campo de edición" (OBJ_EDIT)	ENUM_ALIGN_MODE
OBJPROP_FONTSIZE	Tamaño de caracteres	int
OBJPROP_RAY_LEFT	Rayo va a la izquierda	bool
OBJPROP_RAY_RIGHT	Rayo va a la derecha	bool
OBJPROP_RAY	Línea vertical va a través de todas las ventanas del gráfico	bool
OBJPROP_ELLIPSE	Visualización del elipse entero para el objeto "Arcos de Fibonacci" (OBJ_FIBOARC)	bool
OBJPROP_ARROWCODE	Código de flecha para el objeto "Flecha"	char
OBJPROP_TIMEFRAMES	Visibilidad de objeto en períodos de tiempo	juego de banderas flags

OBJPROP_ANCHOR	Posición del punto de anclaje de un objeto gráfico	ENUM_ARROW_ANCHOR (para OBJ_ARROW), ENUM_ANCHOR_POINT (para OBJ_LABEL, OBJ_BITMAP_LABEL y OBJ_TEXT)
OBJPROP_XDISTANCE	Distancia en píxeles por el eje X desde la esquina de enlace	int
OBJPROP_YDISTANCE	Distancia en píxeles por el eje Y desde la esquina de enlace	int
OBJPROP_DIRECTION	Tendencia del objeto de Gann	ENUM_GANN_DIRECTION
OBJPROP_DEGREE	Nivel de marcación ondulada de Elliott	ENUM_ELLIOT_WAVE_DEGREE
OBJPROP_DRAWLINES	Visualización de líneas para la marcación ondulada de Elliott	bool
OBJPROP_STATE	Estado del botón (Pulsado/ Despulsado)	bool
OBJPROP_CHART_ID	Identificador del objeto "Gráfico" (OBJ_CHART). Permite trabajar con las propiedades de este objeto como con cualquier otro gráfico usando las funciones descritas en el apartado Operaciones con gráficos, pero hay algunas excepciones.	long r/o
OBJPROP_XSIZE	Ширина объекта по оси X в пикселях. Задается для объектов OBJ_LABEL, OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL.	int
OBJPROP_YSIZE	Высота объекта по оси Y в пикселях. Задается для объектов OBJ_LABEL, OBJ_BUTTON, OBJ_CHART, OBJ_BITMAP, OBJ_BITMAP_LABEL, OBJ_EDIT, OBJ_RECTANGLE_LABEL.	int
OBJPROP_XOFFSET	Coordinada X de la esquina superior izquierda del área rectangular de visibilidad en	int

	los objetos gráficos "Etiqueta gráfica" y "Dibujo" (OBJ_BITMAP_LABEL y OBJ_BITMAP). El valor se establece en píxeles respecto a la esquina superior izquierda de la imagen original.	
OBJPROP_YOFFSET	Coordinada Y de la esquina superior izquierda del área rectangular de visibilidad en los objetos gráficos "Etiqueta gráfica" y "Dibujo" (OBJ_BITMAP_LABEL y OBJ_BITMAP). El valor se establece en píxeles respecto a la esquina superior izquierda de la imagen original.	int
OBJPROP_PERIOD	Período de tiempo para el objeto "Gráfico"	ENUM_TIMEFRAMES
OBJPROP_DATE_SCALE	Visualiza la escala de tiempo para el objeto "Gráfico"	bool
OBJPROP_PRICE_SCALE	Visualiza la escala de precios para el objeto "Gráfico"	bool
OBJPROP_CHART_SCALE	Escala para el objeto "Gráfico"	int valores en el rango de 0-5
OBJPROP_BGCOLOR	Color del fondo para el OBJ_EDIT, OBJ_BUTTON, OBJ_RECTANGLE_LABEL	color
OBJPROP_CORNER	Esquina del gráfico para enlazar un objeto gráfico	ENUM_BASE_CORNER
OBJPROP_BORDER_TYPE	Estilo del borde del objeto "Etiqueta rectangular"	ENUM_BORDER_TYPE
OBJPROP_BORDER_COLOR	Color del borde para el objeto OBJ_EDIT y OBJ_BUTTON	color

Cuando se utilizan las operaciones con gráficos para el objeto "Gráfico" (OBJ_CHART), se procede según las siguientes limitaciones:

- no se puede cerrar usando ChartClose();
- no se puede cambiar símbolo/período usando la función ChartSetSymbolPeriod();
- no funcionan las propiedades CHART_SCALE, CHART_BRING_TO_TOP, CHART_SHOW_DATE_SCALE y CHART_SHOW_PRICE_SCALE (ENUM_CHART_PROPERTY_INTEGER).

Para los objetos OBJ_BITMAP_LABEL y OBJ_BITMAP se puede establecer programáticamente el modo especial de visualización de las imágenes. En este modo se muestra sólo aquella parte de la imagen

original a la que se aplica el área rectangular de visibilidad, mientras que el resto de la imagen se hace invisible. El tamaño del área de visibilidad se establece mediante las propiedades OBJPROP_XSIZE y OBJPROP_YSIZE. Usted puede "mover" el área de visibilidad sólo dentro de los márgenes de la imagen original, utilizando las propiedades OBJPROP_XOFFSET y OBJPROP_YOFFSET.

Para las funciones ObjectSetDouble() y ObjectGetDouble()

ENUM_OBJECT_PROPERTY_DOUBLE

Identificador	Descripción	Tipo de la propiedad
OBJPROP_PRICE	Coordinadas de precio	double modificador=número del punto de anclaje
OBJPROP_LEVELVALUE	Valor de nivel	double modificador=número del nivel
OBJPROP_SCALE	Escala (propiedad de objetos de Gann y del objeto "Arcos de Fibonacci")	double
OBJPROP_ANGLE	Ángulo. Para los objetos con el ángulo no especificado, creados desde el programa, el valor es igual a EMPTY_VALUE	double
OBJPROP_DEVIATION	Desviación para el canal de la desviación estándar	double

Para las funciones ObjectSetString() y ObjectGetString()

ENUM_OBJECT_PROPERTY_STRING

Identificador	Descripción	Tipo de la propiedad
OBJPROP_NAME	Nombre de objeto	string
OBJPROP_TEXT	Descripción de objeto (texto contenido en el objeto)	string
OBJPROP_TOOLTIP	El texto de la ayuda emergente. Si la propiedad no está establecida, entonces sale la ayuda generada automáticamente por el terminal. Puede desactivar la visualización de ayuda emergente poniendo el valor "\n" (salto de línea) para esta propiedad	string

OBJPROP_LEVELTEXT	Descripción de nivel	string modificador=número del nivel
OBJPROP_FONT	Fuente	string
OBJPROP_BMPFILE	Nombre del archivo BMP para el objeto "Etiqueta gráfica". Véase también Recursos	string modificador: 0-estado ON, 1-estado OFF
OBJPROP_SYMBOL	Símbolo para el objeto "Gráfico"	string

Para los objetos OBJ_RECTANGLE_LABEL ("Etiqueta rectangular") se puede especificar uno de los tres estilos del borde a los que corresponden los valores de la enumeración ENUM_BORDER_TYPE.

ENUM_BORDER_TYPE

Identificador	Descripción
BORDER_FLAT	Borde plano
BORDER_RAISED	Convexo
BORDER_SUNKEN	Cóncavo

Para el objeto OBJ_EDIT ("Campo de edición") y para la función ChartScreenShot() se puede indicar el tipo de alineación por la horizontal utilizando los valores de la enumeración ENUM_ALIGN_MODE.

ENUM_ALIGN_MODE

Identificador	Descripción
ALIGN_LEFT	Alineación por la izquierda
ALIGN_CENTER	Alineación centrada (sólo para el objeto "Campo de edición")
ALIGN_RIGHT	Alineación derecha

Ejemplo:

```
#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());
      //--- creamos el objeto Label
      ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
      //--- establecemos la coordinada X
      ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
      //--- establecemos la coordinada Y
      ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
      //--- definimos el color del texto
      ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrWhite);
      //--- definimos el texto para el objeto Label
      ObjectSetString(0,label_name,OBJPROP_TEXT,UP);
      //--- definimos la fuente
      ObjectSetString(0,label_name,OBJPROP_FONT,"Wingdings");
      //--- definimos el tamaño de la fuente
      ObjectSetInteger(0,label_name,OBJPROP_FONTSIZE,10);
      //--- rotamos a 45 grados en el sentido de las agujas del reloj
      ObjectSetDouble(0,label_name,OBJPROP_ANGLE,-45);
      //--- deshabilitamos la selección del objeto con el ratón
      ObjectSetInteger(0,label_name,OBJPROP_SELECTABLE,false);
      //--- dibujamos todo eso en el gráfico
      ChartRedraw(0);
     }
  }
```

Modos de enlace de objetos

Los objetos gráficos Text y Label (OBJ_TEXT, OBJ_BITMAP_LABEL y OBJ_LABEL) pueden tener uno de los 9 modos de enlace diferentes de sus coordinadas. Se puede definir una opción necesaria usando la función ObjectSetInteger(handle_de_gráfico, nombre_de_objeto, OBJPROP_ANCHOR, modo_de_enlace), donde modo_de_enlace es uno de los valores de la enumeración ENUM_ANCHOR_POINT.

ENUM_ANCHOR_POINT

Identificador	Descripción
ANCHOR_LEFT_UPPER	Punto de enlace en la esquina superior izquierda
ANCHOR_LEFT	Punto de enlace a la izquierda en el centro
ANCHOR_LEFT_LOWER	Punto de enlace en la esquina inferior izquierda
ANCHOR_LOWER	Punto de enlace abajo en el centro
ANCHOR_RIGHT_LOWER	Punto de enlace en la esquina inferior derecha
ANCHOR_RIGHT	Punto de enlace a la derecha en el centro
ANCHOR_RIGHT_UPPER	Punto de enlace en la esquina superior derecha
ANCHOR_UPPER	Punto de enlace arriba en el centro
ANCHOR_CENTER	Punto de enlace justo en el centro del objeto

Ejemplo:

```
string text_name="my_OBJ_TEXT_object";
if(ObjectFind(0,text_name)<0)
  {
   Print("Object ",text_name," not found. Error code = ",GetLastError());
   //--- obtenemos el precio máximo del gráfico
   double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
   //--- creamos el objeto Label
   ObjectCreate(0,text_name,OBJ_TEXT,0,TimeCurrent(),chart_max_price);
   //--- definimos el color del texto
   ObjectSetInteger(0,text_name,OBJPROP_COLOR,clrWhite);
   //--- definimos el color del fondo
   ObjectSetInteger(0,text_name,OBJPROP_BGCOLOR,clrGreen);
   //--- definimos el texto para el objeto Label
   ObjectSetString(0,text_name,OBJPROP_TEXT,TimeToString(TimeCurrent()));
   //--- definimos la fuente
   ObjectSetString(0,text_name,OBJPROP_FONT,"Trebuchet MS");
   //--- definimos el tamaño la fuente
   ObjectSetInteger(0,text_name,OBJPROP_FONTSIZE,10);
   //--- enlazamos a la esquina superior derecha
   ObjectSetInteger(0,text_name,OBJPROP_ANCHOR,ANCHOR_RIGHT_UPPER);
   //--- rotamos a 90 grados contra el sentido de reloj
```

```
      ObjectSetDouble(0,text_name,OBJPROP_ANGLE,90);
      //--- prohibimos la selección del objeto con el ratón
      ObjectSetInteger(0,text_name,OBJPROP_SELECTABLE,false);
      //--- lo dibujamos en el gráfico
      ChartRedraw(0);
     }
```

Los objetos gráficos Arrow (OBJ_ARROW) tienen sólo 2 opciones de enlazar sus coordinadas. Los identificadores se enumeran en ENUM_ARROW_ANCHOR.

ENUM_ARROW_ANCHOR

Identificador	Descripción
ANCHOR_TOP	Punto de enlace para la flecha se encuentra arriba
ANCHOR_BOTTOM	Punto de enlace para la flecha se encuentra abajo

Ejemplo:

```
void OnStart()
  {
//--- arrays auxiliares
   double Ups[],Downs[];
   datetime Time[];
//--- definimos los arrays como series temporales
   ArraySetAsSeries(Ups,true);
   ArraySetAsSeries(Downs,true);
   ArraySetAsSeries(Time,true);
//--- creamos el manejador del indicador Fractals
   int FractalsHandle=iFractals(NULL,0);
   Print("FractalsHandle = ",FractalsHandle);
//--- ponemos el último error a cero
   ResetLastError();
//--- intentamos copiar valores del indicador
   int copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
   if(copied<=0)
     {
      Print("Fallo al copiar los fractales superiores. Error = ",GetLastError());
      return;
     }

   ResetLastError();
//--- intentamos copiar valores del indicador
   copied=CopyBuffer(FractalsHandle,1,0,1000,Downs);
   if(copied<=0)
     {
      Print("Fallo al copiar los fractales inferiores. Error = ",GetLastError());
      return;
```

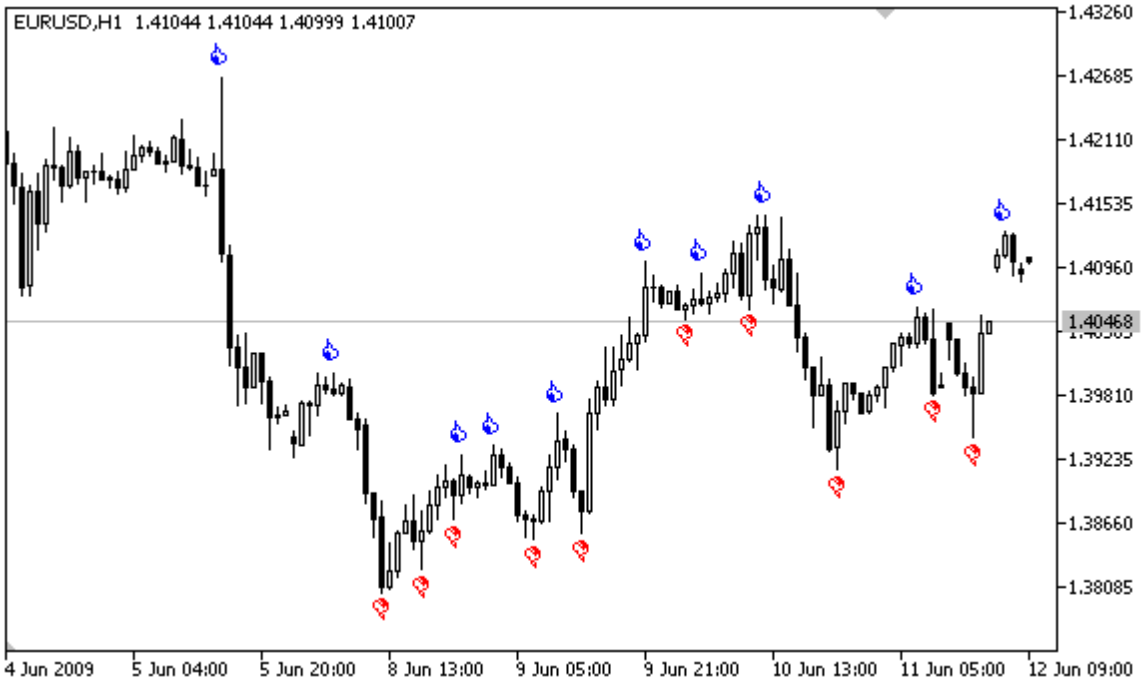
```
     }

   ResetLastError();
//--- copiamos la serie temporal que contiene la hora de apertura de las últimas 1000
   copied=CopyTime(NULL,0,0,1000,Time);
   if(copied<=0)
     {
      Print("Fallo al copiar la hora de apertura de los últimos 1000 barras");
      return;
     }

   int upcounter=0,downcounter=0; // vamos a contar el número de flechas en ellas
   bool created;// vamos a recibir el resultado del intento de creación del objeto
   for(int i=2;i<copied;i++)// repasamos los valores del indicador iFractals
     {
      if(Ups[i]!=EMPTY_VALUE)// encontramos el fractal de arriba
        {
         if(upcounter<10)// creamos no más de 10 objetos "arriba"
           {
            //--- intentamos crear el objeto "arriba"
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_UP,0,Time[i],Ups[i
            if(created)// si ha salido, vamos a tunearlo
              {
               //--- punto de enlace abajo para no pisar la barra
               ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_BOTTOM);
               //--- el último retoque - pintamos
               ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrBlue);
               upcounter++;
              }
           }
        }
      if(Downs[i]!=EMPTY_VALUE)// encontramos el fractal de abajo
        {
         if(downcounter<10)// creamos no más de 10 objetos "abajo"
           {
            //--- intentamos crear el objeto "abajo"
            created=ObjectCreate(0,string(Time[i]),OBJ_ARROW_THUMB_DOWN,0,Time[i],Dow
            if(created)// si lo conseguimos, vamos a tunearlo
              {
               //--- punto de enlace arriba para no pisar la barra
               ObjectSetInteger(0,string(Time[i]),OBJPROP_ANCHOR,ANCHOR_TOP);
               //--- el último retoque - pintamos
               ObjectSetInteger(0,string(Time[i]),OBJPROP_COLOR,clrRed);
               downcounter++;
              }
           }
        }
     }
```

```
}
```

Una vez completado el script, el gráfico será más o menos como se muestra abajo.



Esquina del gráfico a la que se enlaza un objeto

Hay una serie de objetos gráficos para los que podemos definir la esquina del gráfico respecto al cual se especifican las coordinadas en píxeles. So los objetos siguientes (entre paréntesis están indicados los identificadores del tipo de objeto):

- Label (OBJ_LABEL);
- Button (OBJ_BUTTON);
- Chart (OBJ_CHART);
- Bitmap Label (OBJ_BITMAP_LABEL);
- Rectangle Label (OBJ_RECTANGLE_LABEL);
- Edit (OBJ_EDIT).

Para especificar la esquina del gráfico desde la cual se miden las coordinadas X y Y en píxeles, hay que usar la función ObjectSetInteger(chartID, name, OBJPROP_CORNER, **chart_corner**), donde:

- chartID - identificador del gráfico;
- name – nombre del objeto gráfico;
- OBJPROP_CORNER – identificador de la propiedad para determinar la esquina de enlace;
- **chart_corner** – la esquina del gráfico puede adquirir uno de los valores de la enumeración ENUM_BASE_CORNER.

ENUM_BASE_CORNER

Identificador	Descripción
CORNER_LEFT_UPPER	Centro de coordinadas se encuentra en la esquina superior izquierda del gráfico
CORNER_LEFT_LOWER	Centro de coordinadas se encuentra en la esquina inferior izquierda del gráfico
CORNER_RIGHT_LOWER	Centro de coordinadas se encuentra en la esquina inferior derecha del gráfico
CORNER_RIGHT_UPPER	Centro de coordinadas se encuentra en la esquina superior derecha del gráfico

Ejemplo:

```
void CreateLabel(long   chart_id,
                 string name,
                 int    chart_corner,
                 string text_label,
                 int    x_ord,
                 int    y_ord)
  {
//---
   ObjectCreate(chart_id,name,OBJ_LABEL,0,0,0);
   ResetLastError();
   if(!ObjectSetInteger(chart_id,name,OBJPROP_CORNER,chart_corner))
```

```
      Print("Fallo al fijar la esquina de enlace para el objeto ",
            name,", error ",GetLastError());
   ObjectSetInteger(chart_id,name,OBJPROP_XDISTANCE,x_ord);
   ObjectSetInteger(chart_id,name,OBJPROP_YDISTANCE,y_ord);
   ObjectSetString(chart_id,name,OBJPROP_TEXT,text_label);
  }
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//---
   int height=ChartGetInteger(0,CHART_HEIGHT_IN_PIXELS,0);
   int width=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,"0",50,50);
   CreateLabel(0,arrows[1],CORNER_RIGHT_UPPER,"1",50,50);
   CreateLabel(0,arrows[2],CORNER_RIGHT_LOWER,"2",50,50);
   CreateLabel(0,arrows[3],CORNER_LEFT_LOWER,"3",50,50);
  }
```

Visibilidad de objetos

La combinación de banderas de la visibilidad de objeto determina los períodos de tiempo del gráfico en los que se muestra el objeto. Para definir/obtener los valores de la propiedad OBJPROP_TIMEFRAMES podemos usar las funciones ObjectSetInteger()/ObjectGetInteger().

Constante	Valor	Descripción
OBJ_NO_PERIODS	0	El objeto no se muestra en ninguno de los períodos
OBJ_PERIOD_M1	0x00000001	El objeto se dibuja en los gráficos de 1 minuto
OBJ_PERIOD_M2	0x00000002	El objeto se dibuja en los gráficos de 2 minutos
OBJ_PERIOD_M3	0x00000004	El objeto se dibuja en los gráficos de 3 minutos
OBJ_PERIOD_M4	0x00000008	El objeto se dibuja en los gráficos de 4 minutos
OBJ_PERIOD_M5	0x00000010	El objeto se dibuja en los gráficos de 5 minutos
OBJ_PERIOD_M6	0x00000020	El objeto se dibuja en los gráficos de 6 minutos
OBJ_PERIOD_M10	0x00000040	El objeto se dibuja en los gráficos de 10 minutos
OBJ_PERIOD_M12	0x00000080	El objeto se dibuja en los gráficos de 12 minutos
OBJ_PERIOD_M15	0x00000100	El objeto se dibuja en los gráficos de 15 minutos
OBJ_PERIOD_M20	0x00000200	El objeto se dibuja en los gráficos de 20 minutos
OBJ_PERIOD_M30	0x00000400	El objeto se dibuja en los gráficos de 30 minutos
OBJ_PERIOD_H1	0x00000800	El objeto se dibuja en los gráficos de 1 hora
OBJ_PERIOD_H2	0x00001000	El objeto se dibuja en los gráficos de 2 horas
OBJ_PERIOD_H3	0x00002000	El objeto se dibuja en los gráficos de 3 horas
OBJ_PERIOD_H4	0x00004000	El objeto se dibuja en los gráficos de 4 horas
OBJ_PERIOD_H6	0x00008000	El objeto se dibuja en los gráficos de 6 horas

OBJ_PERIOD_H8	0x00010000	El objeto se dibuja en los gráficos de 8 horas
OBJ_PERIOD_H12	0x00020000	El objeto se dibuja en los gráficos de 12 horas
OBJ_PERIOD_D1	0x00040000	El objeto se dibuja en los gráficos de un día
OBJ_PERIOD_W1	0x00080000	El objeto se dibuja en los gráficos semanales
OBJ_PERIOD_MN1	0x00100000	El objeto se dibuja en los gráficos mensuales
OBJ_ALL_PERIODS	0x001fffff	El objeto se dibuja en todos los períodos de tiempo

Las banderas de la visibilidad se puede combinar mediante el símbolo "|", por ejemplo, la combinación de banderas OBJ_PERIOD_M10|OBJ_PERIOD_H4 significa que el objeto será mostrado en los períodos de 10 minutos y 4 horas.

Ejemplo:

```
void OnStart()
  {
//---
   string highlevel="PreviousDayHigh";
   string lowlevel="PreviousDayLow";
   double prevHigh;           // High del día anterior
   double prevLow;            // Low del día anterior
   double highs[],lows[];     // matrices para recibir High y Low

//--- ponemos el último error a cero
   ResetLastError();
//--- obtenemos 2 últimos valores High en período de tiempo diurno
   int highsgot=CopyHigh(Symbol(),PERIOD_D1,0,2,highs);
   if(highsgot>0) // si el copiado ha sido con éxito
     {
      Print("Precios High de los últimos 2 días han sido recibidos con éxito");
      prevHigh=highs[0]; // High del día anterior
      Print("prevHigh = ",prevHigh);
      if(ObjectFind(0,highlevel)<0) // objeto con el nombre highlevel no ha sido enco
        {
         ObjectCreate(0,highlevel,OBJ_HLINE,0,0,0); // creamos el objeto Línea Horizo
        }
      //--- definimos el nivel de precio para la línea highlevel
      ObjectSetDouble(0,highlevel,OBJPROP_PRICE,0,prevHigh);
      //--- establecemos la visibilidad sólo para PERIOD_M10 y PERIOD_H4
      ObjectSetInteger(0,highlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
     }
   else
```

```
      {
       Print("Fallo al recibir los precios High de los últimos 2 días, Error = ",GetLa
      }

//--- ponemos el último error a cero
   ResetLastError();
//--- obtenemos 2 últimos valores Low en período de tiempo diurno
   int lowsgot=CopyLow(Symbol(),PERIOD_D1,0,2,lows);
   if(lowsgot>0) // si el copiado ha sido con éxito
     {
      Print("Precios Low de los últimos 2 días han sido recibidos con éxito");
      prevLow=lows[0]; // Low del día anterior
      Print("prevLow = ",prevLow);
      if(ObjectFind(0,lowlevel)<0) // objeto con el nombre lowlevel no ha sido encont
        {
         ObjectCreate(0,lowlevel,OBJ_HLINE,0,0,0); // creamos el objeto Línea Horizon
        }
      //--- definimos el nivel de precio para la línea lowlevel
      ObjectSetDouble(0,lowlevel,OBJPROP_PRICE,0,prevLow);
      //--- establecemos la visibilidad sólo para  PERIOD_M10 y PERIOD_H4
      ObjectSetInteger(0,lowlevel,OBJPROP_TIMEFRAMES,OBJ_PERIOD_M10|OBJ_PERIOD_H4);
     }
   else Print("Fallo al recibir los precios Low de los últimos 2 días, Error = ",GetL

   ChartRedraw(0); // redibujamos el gráfico por vía forzada
  }
```

Véase también

PeriodSeconds, Period, Períodos de gráficos, Fecha y hora

Niveles de las ondas de Elliott

Las ondas de Elliott están representadas por dos objetos gráficos de los tipos OBJ_ELLIOTWAVE5 y OBJ_ELLIOTWAVE3. Para definir el tamaño de la onda (método de etiquetar las ondas) se usa la propiedad OBJPROP_DEGREE, a la que podemos asignar uno de los valores de la enumeración ENUM_ELLIOT_WAVE_DEGREE.

ENUM_ELLIOT_WAVE_DEGREE

Constante	Descripción
ELLIOTT_GRAND_SUPERCYCLE	Gran superciclo (Grand Supercycle)
ELLIOTT_SUPERCYCLE	Superciclo (Supercycle)
ELLIOTT_CYCLE	Ciclo (Cycle)
ELLIOTT_PRIMARY	Primario (Primary)
ELLIOTT_INTERMEDIATE	Intermedio (Intermediate)
ELLIOTT_MINOR	Menor (Minor)
ELLIOTT_MINUTE	Minute (Minute)
ELLIOTT_MINUETTE	Minuette (Minuette)
ELLIOTT_SUBMINUETTE	Subminuette (Subminuette)

Ejemplo:

```
   for(int i=0;i<ObjectsTotal(0);i++)
     {
      string currobj=ObjectName(0,i);
      if((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE3) ||
         ((ObjectGetInteger(0,currobj,OBJPROP_TYPE)==OBJ_ELLIOTWAVE5)))
        {
         //--- pongamos el nivel de marcación en INTERMEDIATE
         ObjectSetInteger(0,currobj,OBJPROP_DEGREE,ELLIOTT_INTERMEDIATE);
         //--- habilitamos la muestra de las líneas entre las partes superiores de la
         ObjectSetInteger(0,currobj,OBJPROP_DRAWLINES,true);
         //--- definimos el color de las líneas
         ObjectSetInteger(0,currobj,OBJPROP_COLOR,clrBlue);
         //--- definimos el grosor de las líneas
         ObjectSetInteger(0,currobj,OBJPROP_WIDTH,5);
         //--- definimos la descripción
         ObjectSetString(0,currobj,OBJPROP_TEXT,"test script");
        }
     }
```

Objetos de Gann

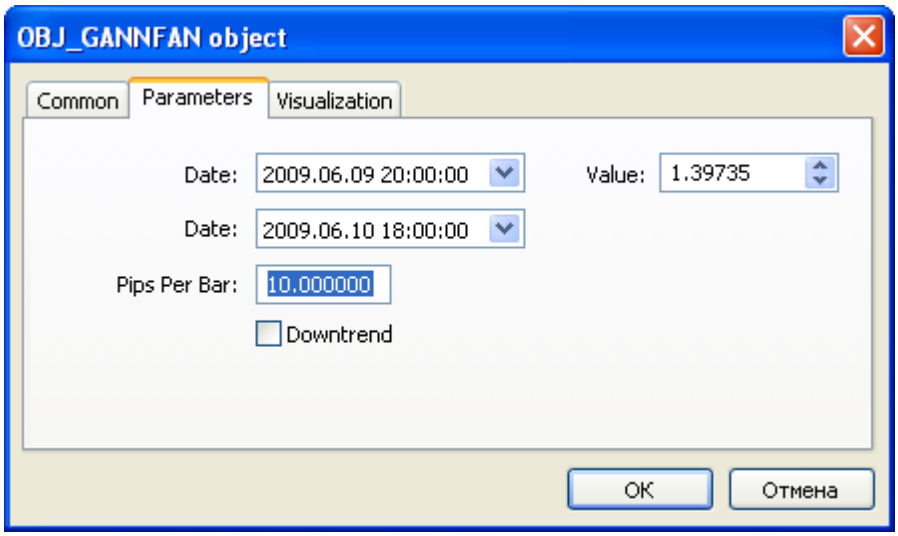
Para los objetos abanico de Gann (OBJ_GANNFAN) y retícula de Gann (OBJ_GANNGRID) podemos especificar uno de dos valores de la enumeración ENUM_GANN_DIRECTION que establece la dirección de tendencia.

ENUM_GANN_DIRECTION

Constante	Descripción
GANN_UP_TREND	Línea de tendencia al alza
GANN_DOWN_TREND	Línea de tendencia a la baja

Para establecer la escala de la línea principar de 1x1 se usa la función ObjectSetDouble(chart_handle, gann_object_name, OBJPROP_SCALE, scale), donde:

- chart_handle - ventana del gráfico en la que se encuentra el objeto;
- gann_object_name - nombre del objeto;
- OBJPROP_SCALE - identificador de la propiedad "Scale";
- scale - escala requerida en unidades Pips/Bar.



Ejemplo de creación del abanico de Gann:

```
void OnStart()
  {
//---
   string my_gann="OBJ_GANNFAN object";
   if(ObjectFind(0,my_gann)<0)// objeto no encontrado
     {
      //--- informamos sobre el fallo
      Print("Object ",my_gann," not found. Error code = ",GetLastError());
      //--- obtenemos el precio máximo del gráfico
      double chart_max_price=ChartGetDouble(0,CHART_PRICE_MAX,0);
      //--- obtenemos el precio mínimo del gráfico
      double chart_min_price=ChartGetDouble(0,CHART_PRICE_MIN,0);
```

```
      //--- ¿Cuántas barras se muestra en el gráfico?
      int bars_on_chart=ChartGetInteger(0,CHART_VISIBLE_BARS);
      //--- creamos una matriz en la que escribimos la hora de apertura de cada barra
      datetime Time[];
      //--- organizamos el acceso a la matriz como en serie temporal
      ArraySetAsSeries(Time,true);
      //--- ahora copiamos en él los datos de barras visibles en el gráfico
      int times=CopyTime(NULL,0,0,bars_on_chart,Time);
      if(times<=0)
        {
         Print("Fallo al copiar un array con la hora de apertura!");
         return;
        }
      //--- preparativos preliminares finalizados

      //--- índice de la barra central en el gráfico
      int center_bar=bars_on_chart/2;
      //--- ecuador del gráfico - entre el máximo y el mínimo
      double mean=(chart_max_price+chart_min_price)/2.0;
      //--- establecemos las coordinadas del primer punto de enlace en el centro
      ObjectCreate(0,my_gann,OBJ_GANNFAN,0,Time[center_bar],mean,
                   //--- el segundo punto de enlace a la derecha
                   Time[center_bar/2],(mean+chart_min_price)/2.0);
      Print("Time[center_bar] = "+(string)Time[center_bar]+"  Time[center_bar/2] = "+
      //Print("Time[center_bar]/="+Time[center_bar]+"  Time[center_bar/2]="+Time[cent
      //--- establecemos la escala en unidades Pips/Bar
      ObjectSetDouble(0,my_gann,OBJPROP_SCALE,10);
      //--- definimos la tendencia de la línea
      ObjectSetInteger(0,my_gann,OBJPROP_DIRECTION,GANN_UP_TREND);
      //--- definimos el grosor de la línea
      ObjectSetInteger(0,my_gann,OBJPROP_WIDTH,1);
      //--- definimos el estilo de la línea
      ObjectSetInteger(0,my_gann,OBJPROP_STYLE,STYLE_DASHDOT);
      //--- y el color de la línea
      ObjectSetInteger(0,my_gann,OBJPROP_COLOR,clrYellowGreen);
      //--- permitimos al usuario seleccionar el objeto
      ObjectSetInteger(0,my_gann,OBJPROP_SELECTABLE,true);
      //--- lo seleccionamos nosotros mismos
      ObjectSetInteger(0,my_gann,OBJPROP_SELECTED,true);
      //--- redibujamos el gráfico
      ChartRedraw(0);
     }
  }
```

Colores Web

La siguientes constantes de colores están predefinidas para el tipo color:

						clrDarkTurquoise	
clrLightSeaGreen							
clrGoldenrod	clrMediumSpringGreen	clrLawnGreen					
	clrOrange	clrGold	clrYellow	clrChartreuse	clrLime	clrSpringGreen	clrAqua
clrDeepSkyBlue		clrMagenta					
		clrMediumTurquoise		clrTurquoise			clrDarkKhaki
		clrGreenYellow	clrMediumAquamarine				clrOrchid
clrMediumPurple				clrDarkGray	clrSandyBrown		clrTan
	clrBurlyWood	clrHotPink		clrViolet		clrSkyBlue	clrLightSalmon
clrPlum	clrKhaki	clrLightGreen	clrAquamarine	clrSilver	clrLightSkyBlue	clrLightSteelBlue	clrLightBlue
clrPaleGreen	clrThistle	clrPowderBlue	clrPaleGoldenrod	clrPaleTurquoise	clrLightGray	clrWheat	clrNavajoWhite
clrMoccasin	clrLightPink	clrGainsboro	clrPeachPuff	clrPink	clrBisque	clrLightGoldenrod	clrBlanchedAlmond
clrLemonChiffon	clrBeige	clrAntiqueWhite	clrPapayaWhip	clrCornsilk	clrLightYellow	clrLightCyan	clrLinen
clrLavender	clrMistyRose	clrOldLace	clrWhiteSmoke	clrSeashell	clrIvory	clrHoneydew	clrAliceBlue
clrLavenderBlush	clrMintCream	clrSnow	clrWhite				

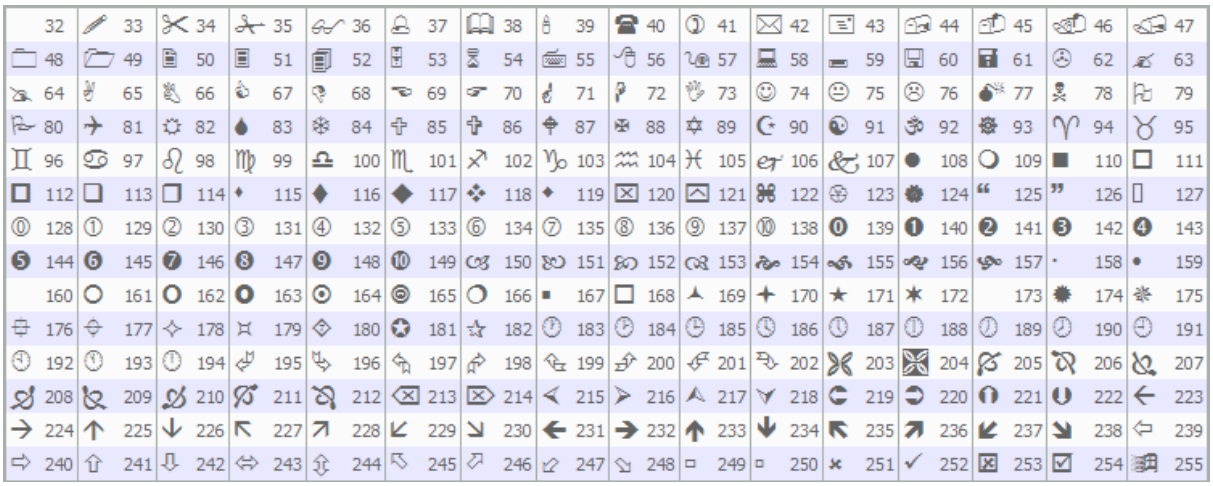
Podemos definir el color para un objeto usando la función ObjectSetInteger(), y para indicadores personalizados usando la función PlotIndexSetInteger(). Las funciones similares ObjectGetInteger() y PlotIndexGetInteger() sirven para obtener el valor de un color.

Ejemplo:

```
//---- indicator settings
#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   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

Estos son los caracteres de la fuente Wingdings que se usan con el objeto OBJ_ARROW:



El símbolo necesario se establece usando la función ObjectSetInteger().

Ejemplo:

```
void OnStart()
  {
//---
   string up_arrow="up_arrow";
   datetime time=TimeCurrent();
   double lastClose[1];
   int close=CopyClose(Symbol(),Period(),0,1,lastClose);        // obtenemos el preci
//--- si hemos recibido el precio
   if(close>0)
     {
      ObjectCreate(0,up_arrow,OBJ_ARROW,0,0,0,0,0);           // creamos la flecha
      ObjectSetInteger(0,up_arrow,OBJPROP_ARROWCODE,241);     // definimos el código d
      ObjectSetInteger(0,up_arrow,OBJPROP_TIME,time);         // definimos la hora
      ObjectSetDouble(0,up_arrow,OBJPROP_PRICE,lastClose[0]);// definimos el precio
      ChartRedraw(0);                                         // redibujamos la ventan
     }
   else
      Print("Fallo al recibir el último precio Close!");
  }
```

Constantes de indicadores

Hay 37 constantes predefinidas de indicadores técnicos estándares que se puede utilizar en los programas en el lenguaje MQL5. Además, existe la posibilidad de crear sus propios indicadores personalizados mediante la función iCustom(). Todas las constantes necesarias para eso están divididas en 5 grupos:

- Constantes de precio – para seleccionar el tipo de precio o volumen por los cuales se calcula el indicador;
- Métodos de alisamiento – métodos buil-in de alisamiento utilizados en los indicadores;
- Líneas de indicadores – identificadores de buffers de indicadores durante el acceso a los valores de indicadores usando la función CopyBuffer();
- Estilos de dibujo – para especificar uno de los 18 tipos de dibujo y para establecer el estilo de dibujo de la línea;
- Propiedades de indicadores personalizados – se usa en las funciones para trabajar con los indicadores personalizados;
- Tipos de indicadores – sirven para especificar el tipo de indicador técnico a la hora de crear un manejador (handle) a través de la función IndicatorCreate();
- Identificadores de tipos de datos - se usan para establecer el tipo de datos traspasados por el array del tipo MqlParam en la función IndicatorCreate().

Constantes de precios

Los indicadores técnicos requieren para sus cálculos la especificación de los valores de los precios y/o volúmenes a base de los cuales serán calculados. Hay 7 identificadores predefinidos de la enumeración ENUM_APPLIED_PRICE que se usan para especificar la base de precios necesaria para los cálculos.

ENUM_APPLIED_PRICE

Identificador	Descripción
PRICE_CLOSE	Precio de cierre
PRICE_OPEN	Precio de apertura
PRICE_HIGH	Precio máximo en el período
PRICE_LOW	Precio mínimo en el período
PRICE_MEDIAN	Precio mediano, (high+low)/2
PRICE_TYPICAL	Precio típico, (high+low+close)/3
PRICE_WEIGHTED	Precio promedio, (high+low+close+close)/4

Si en los cálculos se usa el volumen, hay que indicar uno de dos valores de la enumeración ENUM_APPLIED_VOLUME.

ENUM_APPLIED_VOLUME

Identificador	Descripción
VOLUME_TICK	Volumen de tick
VOLUME_REAL	Volumen comercial

El indicador técnico iStochastic() se calcula de dos maneras, utilizando:

- o sólo los precios Close;
- o los precios High y Low.

Para eligir la opción necesaria de cálculo hay que indicar uno de los valores de la enumeración ENUM_STO_PRICE.

ENUM_STO_PRICE

Identificador	Descripción
STO_LOWHIGH	Calcular basándose en los precios Low/High
STO_CLOSECLOSE	Calcular basándose en los precios Close/Close

Si un indicador técnico para sus cálculos utiliza los datos cuyo tipo es definido por la enumeración ENUM_APPLIED_PRICE, entonces el manejador de cualquier indicador (built-in en el terminal o escrito por el usuario) puede ser usado como serie de precios de entrada. En este caso para los cálculos serán utilizados los valores del buffer cero del indicador. Eso permite desarrollar fácilmente los valores de un indicador a base de los valores del otro. El manejador de un indicador personalizado se crea llamando a la función iCustom().

Ejemplo:

```
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
//--- input parameters
input int      RSIperiod=14;         // período para calcular RSI
input int      Smooth=8;             // período de alisamiento RSI
input ENUM_MA_METHOD meth=MODE_SMMA; // método de alisamiento
//---- plot RSI
#property indicator_label1  "RSI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//---- plot RSI_Smoothed
#property indicator_label2  "RSI_Smoothed"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrNavy
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- indicator buffers
double         RSIBuffer[];          // aquí vamos a almacenar los valores RSI
double         RSI_SmoothedBuffer[]; // aquí estarán los valores suavizados RSI
int            RSIhandle;            // descriptor de indicador RSI
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,RSIBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,RSI_SmoothedBuffer,INDICATOR_DATA);
   IndicatorSetString(INDICATOR_SHORTNAME,"iRSI");
   IndicatorSetInteger(INDICATOR_DIGITS,2);
//---
   RSIhandle=iRSI(NULL,0,RSIperiod,PRICE_CLOSE);
//---
  }
//+------------------------------------------------------------------+
//| Custom indicator iteration function                              |
//+------------------------------------------------------------------+
int OnCalculate(const int rates_total,
                const int prev_calculated,
                const int begin,
                const double &price[]
                )

  {
```

```
//--- ponemos a cero el valor del último error
   ResetLastError();
//--- obtenemos los datos del indicador RSI en un array RSIBuffer[]
   int copied=CopyBuffer(RSIhandle,0,0,rates_total,RSIBuffer);
   if(copied<=0)
     {
      Print("Fallo al copiar los valores del indicador RSI. Error = ",
            GetLastError(),",  copied = ",copied);
      return(0);
     }
//--- creamos el indicador de la media usando los valores del indicador RSI
   int RSI_MA_handle=iMA(NULL,0,Smooth,0,meth,RSIhandle);
   copied=CopyBuffer(RSI_MA_handle,0,0,rates_total,RSI_SmoothedBuffer);
   if(copied<=0)
     {
      Print("Fallo al copiar el indicador suavizado RSI. Error = ",
            GetLastError(),",  copied = ",copied);
      return(0);
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Métodos de alisamiento

Muchos indicadores técnicos se basan sobre unos u otros métodos de alisamiento de las series de precios. Algunos indicadores técnicos estándares requieren la especificación del tipo de alisamiento como un parámetro de entrada. Los identificadores de la enumeración ENUM_MA_METHOD sirven para especificar el tipo deseado de alisamiento.

ENUM_MA_METHOD

Identificador	Descripción
MODE_SMA	Media móvil simple
MODE_EMA	Media móvil exponencial
MODE_SMMA	Media móvil suavizada
MODE_LWMA	Media móvil pronderada

Ejemplo:

```
double ExtJaws[];
double ExtTeeth[];
double ExtLips[];
//---- handles for moving averages
int    ExtJawsHandle;
int    ExtTeethHandle;
int    ExtLipsHandle;
//--- get MA's handles
ExtJawsHandle=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtTeethHandle=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN);
ExtLipsHandle=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN);
```

Líneas de indicadores

Algunos indicadores técnicos tienen varios buffers dibujados en el gráfico. La numeración de los buffers de indicadores se empieza desde 0. A la hora de copiar los valores del indicador en un array del tipo double a través de la función CopyBuffer(), para algunos indicadores podemos indicar el identificador de un búfer copiado en vez de su número.

Identificadores de las líneas de indicadores admisibles a la hora de copiar los valores de indicadores iMACD(), iRVI() y iStochastic()

Constante	Valor	Descripción
MAIN_LINE	0	Línea principal
SIGNAL_LINE	1	Línea de señales

Identificadores de las líneas de indicadores admisibles a la hora de copiar los valores de indicadores ADX() y ADXW()

Constante	Valor	Descripción
MAIN_LINE	0	Línea principal
PLUSDI_LINE	1	Línea +DI
MINUSDI_LINE	2	Línea -DI

Identificadores de las líneas de indicadores admisibles a la hora de copiar los valores de indicador iBands()

Constante	Valor	Descripción
BASE_LINE	0	Línea principal
UPPER_BAND	1	Límite superior
LOWER_BAND	2	Límite inferior

Identificadores de las líneas de indicadores admisibles a la hora de copiar los valores de indicadores iEnvelopes() y iFractals()

Constante	Valor	Descripción
UPPER_LINE	0	Línea superior
LOWER_LINE	1	Línea inferior

Identificadores de las líneas de indicadores admisibles a la hora de copiar los valores de indicador iGator()

Constante	Valor	Descripción
UPPER_HISTOGRAM	0	Histograma de arriba
LOWER_HISTOGRAM	2	Histograma de abajo

Identificadores de las líneas de indicadores admisibles a la hora de copiar los valores de indicador iAlligator()

Constante	Valor	Descripción
GATORJAW_LINE	0	Línea de mandíbulas
GATORTEETH_LINE	1	Línea de dientes
GATORLIPS_LINE	2	Línea de labios

Identificadores de las líneas de indicadores admisibles a la hora de copiar los valores de indicador iIchimoku()

Constante	Valor	Descripción
TENKANSEN_LINE	0	Línea Tenkan-sen
KIJUNSEN_LINE	1	Línea Kijun-sen
SENKOUSPANA_LINE	2	Línea Senkou Span A
SENKOUSPANB_LINE	3	Línea Senkou Span B
CHINKOUSPAN_LINE	4	Línea Chinkou Span

Estilos de dibujo

Cuando creamos un indicador personalizado se puede indicar uno de 18 tipos de construcción gráfica (modo de visualizar en la ventana principal o subventana del gráfico), cuyos valores se especifican en la enumeración ENUM_DRAW_TYPE.

En un indicador personalizado se permite usar cualquier tipo de construcción/dibujo de indicadores. Cada tipo de construcción requiere que se indique de uno a cinco arrays globales para almacenar los datos necesarios para dibujar. Hay que enlazar estas matrices de datos con los buffers de indicadores mediante la función SetIndexBuffer(), y especificar el tipo de datos de la enumeración ENUM_INDEXBUFFER_TYPE para cada buffer.

Dependiendo del estilo de dibujo, podemos necesitar de uno a cuatro buffers de valores (marcados como INDICATOR_DATA). Si un estilo admite la alternación dinámica de colores (todos los colores contienen la palabra COLOR en sus nombres), entonces necesitamos un buffer de color (tipo indicado INDICATOR_COLOR_INDEX). El buffer de colores siempre se enlaza después de los buffers de valores que corresponden al estilo.

ENUM_DRAW_TYPE

Identificador	Descripción	Buffer de valores	Buffer de color
DRAW_NONE	No se dibuja	1	0
DRAW_LINE	Línea	1	0
DRAW_SECTION	Sección	1	0
DRAW_HISTOGRAM	Histograma de la línea cero	1	0
DRAW_HISTOGRAM2	Histograma de dos buffers de indicadores	2	0
DRAW_ARROW	Dibujo de flechas	1	0
DRAW_ZIGZAG	Estilo Zigzag permite secciones verticales en la barra	2	0
DRAW_FILLING	Relleno de color entre dos niveles	2	0
DRAW_BARS	Visualización como secuencia de barras	4	0
DRAW_CANDLES	Visualización como secuencia de velas	4	0
DRAW_COLOR_LINE	Línea multicolor	1	1
DRAW_COLOR_SECTION	Secciones multicolor	1	1
DRAW_COLOR_HISTOGRAM	Histograma multicolor desde la línea cero	1	1

DRAW_COLOR_HISTOGRAM2	Histograma multicolor de dos buffers de indicadores	2	1
DRAW_COLOR_ARROW	Dibujo con flechas multicolor	1	1
DRAW_COLOR_ZIGZAG	ZigZag multicolor	2	1
DRAW_COLOR_BARS	Barras multicolor	4	1
DRAW_COLOR_CANDLES	Velas multicolor	4	1

Para perfeccionar la visualización del tipo seleccionado de dibujo podemos usar los identificadores que vienen en la enumeración ENUM_PLOT_PROPERTY.

Para las funciones PlotIndexSetInteger() y PlotIndexGetInteger()

ENUM_PLOT_PROPERTY_INTEGER

Identificador	Descripción	Tipo de la propiedad
PLOT_ARROW	Código de flecha para el estilo DRAW_ARROW	uchar
PLOT_ARROW_SHIFT	Desplazamiento vertical de flechas para el estilo DRAW_ARROW	int
PLOT_DRAW_BEGIN	Número de barras iniciales sin dibujar y valores en DataWindow	int
PLOT_DRAW_TYPE	Tipo de construcción gráfica	ENUM_DRAW_TYPE
PLOT_SHOW_DATA	Indica la visualización de valores de construcción en la ventana DataWindow	bool
PLOT_SHIFT	Desplazamiento de representación gráfica del indicador respecto al eje de tiempo en barras	int
PLOT_LINE_STYLE	Estilo de la línea de dibujo	ENUM_LINE_STYLE
PLOT_LINE_WIDTH	Grosor de línea de dibujo	int
PLOT_COLOR_INDEXES	Número de colores	int
PLOT_LINE_COLOR	Índice del buffer que contiene el color	color modificador=número del índice de color

Para la función PlotIndexSetDouble()

ENUM_PLOT_PROPERTY_DOUBLE

Identificador	Descripción	Tipo de la propiedad
PLOT_EMPTY_VALUE	Valor vacío para una representación gráfica, para la que no hay dibujo	double

Para la función PlotIndexSetString()

ENUM_PLOT_PROPERTY_STRING

Identificador	Descripción	Tipo de la propiedad
PLOT_LABEL	Nombre de la serie gráfica de indicador para la visualización en DataWindow. Para los estilos gráficos complejos, que requieren varios búferes de indicador para su visualización, los nombres para cada búfer se puede establecer utilizando ";" como separador. El ejemplo del código se puede encontrar en DRAW_CANDLES	string

Hay 5 estilos que se usan para dibujar una línea en un indicador personalizado. Se puede usarlas sólo si tienen el grosor de 0 a 1.

ENUM_LINE_STYLE

Identificador	Descripción
STYLE_SOLID	Línea continua
STYLE_DASH	Polilínea
STYLE_DOT	Línea de puntos
STYLE_DASHDOT	Línea de punto y raya
STYLE_DASHDOTDOT	Línea de dos puntos y raya

Para definir el estilo de dibujo de línea y el tipo de dibujo, se usa la función PlotIndexSetInteger(). A través de la función ObjectSetInteger() se puede determinar el espesor y el estilo del dibujo de niveles para las extensiones de Fibonacci.

Ejemplo:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- indicator buffers
```

```
double         MABuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- enlace del array con el búfer de indicador con el índice 0
   SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
//--- establecer el dibujo de línea
   PlotIndexSetInteger(0,PLOT_DRAW_TYPE,DRAW_LINE);
//--- configuración del estilo para dibujar la línea
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,STYLE_DOT);
//--- color de línea
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrRed);
//--- espesor de líneas
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
//--- etiqueta para la línea
   PlotIndexSetString(0,PLOT_LABEL,"Moving Average");
//---
  }
//+------------------------------------------------------------------+
//| 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(int i=prev_calculated;i<rates_total;i++)
     {
      MABuffer[i]=close[i];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Propiedades de indicadores personalizados

El número de los buffers de indicadores que se puede usar en un indicador personalizado no está limitado. Pero para cada array que se designa como búfer de indicador a través de la función SetIndexBuffer() es necesario indicar el tipo de datos que él va a almacenar. Puede ser uno de los valores de la enumeración ENUM_INDEXBUFFER_TYPE.

ENUM_INDEXBUFFER_TYPE

Identificador	Descripción
INDICATOR_DATA	Datos para dibujar
INDICATOR_COLOR_INDEX	Colores
INDICATOR_CALCULATIONS	Buffers auxiliares para los cálculos intermedios

Un indicador personalizado cuenta con una variedad de configuraciones para proporcionar una visualización y percepción convenientes. Estas configuraciones se realizan a través de asignación de las propiedades de indicador correspondientes usando las funciones IndicatorSetDouble(), IndicatorSetInteger() y IndicatorSetString(). Los identificadores de las propiedades del indicador se encuentran en la enumeración ENUM_CUSTOMIND_PROPERTY.

ENUM_CUSTOMIND_PROPERTY_INTEGER

Identificador	Descripción	Tipo de la propiedad
INDICATOR_DIGITS	Precisión de dibujo de los valores del indicador	int
INDICATOR_HEIGHT	El alto fijo de la propia ventana del indicador (comando del preprocesador #property indicator_height)	int
INDICATOR_LEVELS	Número de niveles en la ventana del indicador	int
INDICATOR_LEVELCOLOR	Color de la línea del nivel	color modificador - número de nivel
INDICATOR_LEVELSTYLE	Estilo de la línea del nivel	ENUM_LINE_STYLE modificador - número de nivel
INDICATOR_LEVELWIDTH	Grosor de la línea del nivel	int modificador - número de nivel

ENUM_CUSTOMIND_PROPERTY_DOUBLE

Identificador	Descripción	Tipo de la propiedad
INDICATOR_MINIMUM	Mínimo de la ventana del indicador	double
INDICATOR_MAXIMUM	Máximo de la ventana del indicador	double

INDICATOR_LEVELVALUE	Valor del nivel	double modificador - número de nivel

ENUM_CUSTOMIND_PROPERTY_STRING

Identificador	Descripción	Tipo de la propiedad
INDICATOR_SHORTNAME	Nombre breve del indicador	string
INDICATOR_LEVELTEXT	Descripción del nivel	string modificador - número de nivel

Ejemplos:

```
//--- indicator settings
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
#property indicator_type1   DRAW_LINE
#property indicator_type2   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_color2  clrRed
//--- input parameters
extern int KPeriod=5;
extern int DPeriod=3;
extern int Slowing=3;
//--- indicator buffers
double MainBuffer[];
double SignalBuffer[];
double HighesBuffer[];
double LowesBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,MainBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,HighesBuffer,INDICATOR_CALCULATIONS);
   SetIndexBuffer(3,LowesBuffer,INDICATOR_CALCULATIONS);
//--- set accuracy
   IndicatorSetInteger(INDICATOR_DIGITS,2);
//--- set levels
   IndicatorSetInteger(INDICATOR_LEVELS,2);
   IndicatorSetDouble(INDICATOR_LEVELVALUE,0,20);
   IndicatorSetDouble(INDICATOR_LEVELVALUE,1,80);
//--- set maximum and minimum for subwindow
   IndicatorSetDouble(INDICATOR_MINIMUM,0);
   IndicatorSetDouble(INDICATOR_MAXIMUM,100);
```

```
//--- sets first bar from what index will be drawn
   PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,KPeriod+Slowing-2);
   PlotIndexSetInteger(1,PLOT_DRAW_BEGIN,KPeriod+Slowing+DPeriod);
//--- set style STYLE_DOT for second line
   PlotIndexSetInteger(1,PLOT_LINE_STYLE,STYLE_DOT);
//--- name for DataWindow and indicator subwindow label
   IndicatorSetString(INDICATOR_SHORTNAME,"Stoch("+KPeriod+","+DPeriod+","+Slowing+")
   PlotIndexSetString(0,PLOT_LABEL,"Main");
   PlotIndexSetString(1,PLOT_LABEL,"Signal");
//--- sets drawing line empty value
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
   PlotIndexSetDouble(1,PLOT_EMPTY_VALUE,0.0);
//--- initialization done
  }
```

Tipos de indicadores técnicos

Hay dos opciones de crear de forma programada un manejador del indicador para el acceso posterior a sus valores. El primer modo consiste en indicar directamente el nombre de la función de la lista de indicadores técnicos. El segundo modo permite, usando la función IndicatorCreate(), crear uniformemente un manejador de cualquier indicador asignando un identificador de la enumeración ENUM_INDICATOR. Ambas opciones valen para crear un manejador, podemos usar la opción que nos parezca más apropiada en cada caso particular durante el desarrollo de un programa en MQL5.

Cuando se crea un indicador del tipo IND_CUSTOM, el campo *type* del primer elemento de un array de parámetros de entrada MqlParam debe tener obligatoriamente el valor TYPE_STRING de la enumeración ENUM_DATATYPE, mientras que el campo *string_value* del primer elemento debe tener el nombre del indicador personalizado.

ENUM_INDICATOR

Identificador	Indicador
IND_AC	Accelerator Oscillator
IND_AD	Accumulation/Distribution
IND_ADX	Average Directional Index
IND_ADXW	ADX by Welles Wilder
IND_ALLIGATOR	Alligator
IND_AMA	Adaptive Moving Average
IND_AO	Awesome Oscillator
IND_ATR	Average True Range
IND_BANDS	Bollinger Bands®
IND_BEARS	Bears Power
IND_BULLS	Bulls Power
IND_BWMFI	Market Facilitation Index
IND_CCI	Commodity Channel Index
IND_CHAIKIN	Chaikin Oscillator
IND_CUSTOM	Custom indicator
IND_DEMA	Double Exponential Moving Average
IND_DEMARKER	DeMarker
IND_ENVELOPES	Envelopes
IND_FORCE	Force Index
IND_FRACTALS	Fractals
IND_FRAMA	Fractal Adaptive Moving Average
IND_GATOR	Gator Oscillator

IND_ICHIMOKU	Ichimoku Kinko Hyo
IND_MA	Moving Average
IND_MACD	MACD
IND_MFI	Money Flow Index
IND_MOMENTUM	Momentum
IND_OBV	On Balance Volume
IND_OSMA	OsMA
IND_RSI	Relative Strength Index
IND_RVI	Relative Vigor Index
IND_SAR	Parabolic SAR
IND_STDDEV	Standard Deviation
IND_STOCHASTIC	Stochastic Oscillator
IND_TEMA	Triple Exponential Moving Average
IND_TRIX	Triple Exponential Moving Averages Oscillator
IND_VIDYA	Variable Index Dynamic Average
IND_VOLUMES	Volumes
IND_WPR	Williams' Percent Range

Identificadores de tipos de datos

Cuando se crea un manejador del indicador a través de la función IndicatorCreate(), un array del tipo MqlParam tiene que ser especificado como el último parámetro. Como consecuencia, la estructura MqlParam, que describe los parámetros del indicador, contiene un campo especial *type*. Este campo contiene la información sobre el tipo de datos (tipo real, de números enteros o de cadena), los que se pasan a los elementos concretos de este array. El valor de este campo de la estructura MqlParam puede ser uno de los valores de la enumeración ENUM_DATATYPE.

ENUM_DATATYPE

Identificador	Tipo de datos
TYPE_BOOL	bool
TYPE_CHAR	char
TYPE_UCHAR	uchar
TYPE_SHORT	short
TYPE_USHORT	ushort
TYPE_COLOR	color
TYPE_INT	int
TYPE_UINT	uint
TYPE_DATETIME	datetime
TYPE_LONG	long
TYPE_ULONG	ulong
TYPE_FLOAT	float
TYPE_DOUBLE	double
TYPE_STRING	string

Cada elemento de este array describe un parámetro de entrada correspondiente de un indicador técnico que se crea, por eso el tipo y el orden de los elementos en el array tiene que ser mantenido estrictamente de acuerdo con la descripción.

Estado de entorno

Las constantes que describen el entorno corriente de ejecución de un programa mql5 se dividen en dos grupos:

- Estado del terminal de cliente – información sobre el terminal de cliente;
- Información sobre el programa MQL5 en ejecución – propiedades de un programa mql5 que ayudan a dirigir adicionalmente su comportamiento;
- Información sobre el instrumento – obtención de información comercial sobre el instrumento;
- Información sobre la cuenta – información sobre la cuenta comercial corriente;
- Estadística de simulación – resultados de prueba del Asesor Experto.

Estado del terminal de cliente

La información sobre el terminal de cliente se obtiene a través de estas dos funciones: TerminalInfoInteger() y TerminalInfoString(). Para parámetros, estas funciones aceptan valores de las enumeraciones ENUM_TERMINAL_INFO_INTEGER y ENUM_TERMINAL_INFO_STRING respectivamente.

ENUM_TERMINAL_INFO_INTEGER

Identificador	Descripción	Tipo de la propiedad
TERMINAL_BUILD	Número de la build del terminal	int
TERMINAL_CONNECTED	Conexión al servidor comercial	bool
TERMINAL_DLLS_ALLOWED	Permiso de usar DLL	bool
TERMINAL_TRADE_ALLOWED	Permiso de hacer comercio	bool
TERMINAL_EMAIL_ENABLED	Permiso de enviar correo electrónico usando el servidor SMTP y nombre de usuario especificados en las configuraciones del terminal	bool
TERMINAL_FTP_ENABLED	Permiso de enviar informes a través de FTP a un servidor indicado para una cuenta comercial especificada en las configuraciones del terminal	bool
TERMINAL_MAXBARS	Número máximo de barras en el gráfico	int
TERMINAL_CODEPAGE	Número de la página de código del idioma instalado en el terminal de cliente	int

Las operaciones de archivos pueden realizarse sólo en dos directorios; las rutas correspondientes se puede obtener llamando a las propiedades de TERMINAL_DATA_PATH y TERMINAL_COMMONDATA_PATH.

ENUM_TERMINAL_INFO_STRING

Identificador	Descripción	Tipo de la propiedad
TERMINAL_LANGUAGE	Idioma del terminal	string
TERMINAL_COMPANY	Nombre de la empresa	string
TERMINAL_NAME	Nombre del terminal	string
TERMINAL_PATH	Carpeta de la que se inicia el terminal	string
TERMINAL_DATA_PATH	Carpeta donde se almacenan los datos del terminal	string

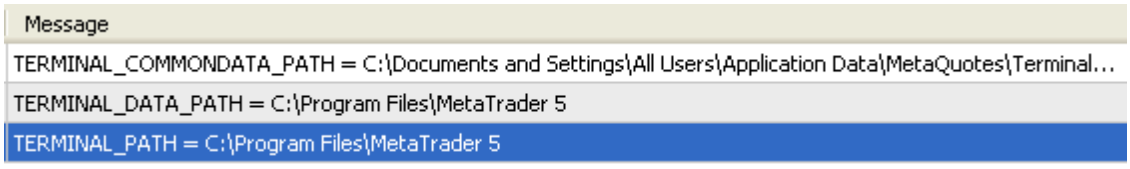
TERMINAL_COMMONDATA_PATH	Carpeta general de todos los terminales de cliente instalados en el ordenador	string

Para mejor entendimiento de las rutas que se guardan en las propiedades de los parámetros TERMINAL_PATH, TERMINAL_DATA_PATH y TERMINAL_COMMONDATA_PATH, se recomienda ejecutar el script que devolverá estos valores para la copia del terminal instalada en su ordenador.

Ejemplo: Script devuelve la información sobre las rutas del terminal de cliente

```
//+------------------------------------------------------------------+
//|                                          Check_TerminalPaths.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://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_COMMONDATA_PATH));
  }
```

Como resultado de su ejecución, en el Registro de Asesores Expertos aparecerán los mensajes parecidos a los que vienen más abajo.



Información sobre el programa MQL5 en ejecución

Para obtener la información sobre los programas mql5 que actualmente están funcionando, se usan las constantes de las enumeraciones ENUM_MQL5_INFO_INTEGER y ENUM_MQL5_INFO_STRING.

Para la función MQL5InfoInteger()

ENUM_MQL5_INFO_INTEGER

Identificador	Descripción	Tipo de la propiedad
MQL5_PROGRAM_TYPE	Tipo del programa mql5	ENUM_PROGRAM_TYPE
MQL5_DLLS_ALLOWED	Permiso de usar DLL para **este programa iniciado**	bool
MQL5_TRADE_ALLOWED	Permiso para tradear concedido a **este programa iniciado**	bool
MQL5_DEBUG	Indica que el programa iniciado trabaja en el modo de depuración	bool
MQL5_PROFILER	Indica que el programa iniciado trabaja en el modo de perfiladura del código	bool
MQL5_TESTER	Indica que el programa iniciado trabaja en el Probador	bool
MQL5_OPTIMIZATION	Indica que el programa iniciado trabaja en el proceso de optimización	bool
MQL5_VISUAL_MODE	Indica que el programa iniciado trabaja en el modo visual de simulación	bool
MQL5_FRAME_MODE	Indica que el EA iniciado en el gráfico trabaja en el modo de recogida de los frames de resultados de la optimización	bool
MQL5_LICENSE_TYPE	Tipo de licencia del módulo EX5. La licencia se refiere al módulo EX5 desde el cual se hace la solicitud con el uso de Mql5InfoInteger (MQL5_LICENSE_TYPE).	ENUM_LICENSE_TYPE

Para la función MQL5InfoString()

ENUM_MQL5_INFO_STRING

Identificador	Descripción	Tipo de la propiedad
MQL5_PROGRAM_NAME	Nombre del programa mql5 en ejecución	string
MQL5_PROGRAM_PATH	Ruta para **dicho programa en ejecución**	string

Para obtener la información sobre el tipo del programa en ejecución podemos usar los valores de la enumeración ENUM_PROGRAM_TYPE.

ENUM_PROGRAM_TYPE

Identificador	Descripción
PROGRAM_SCRIPT	Script
PROGRAM_EXPERT	Experto
PROGRAM_INDICATOR	Indicador

ENUM_LICENSE_TYPE

Identificador	Descripción
LICENSE_FREE	Versión gratuita ilimitada
LICENSE_DEMO	Versión demo del producto de pago de la Tienda. Funciona sólo en el Probador de Estrategias
LICENSE_FULL	La versión de licencia que ha sido comprada admite sólo tres activaciones
LICENSE_TIME	Limitación en el tiempo de trabajo

Ejemplo:

```
ENUM_PROGRAM_TYPE mql_program=(ENUM_PROGRAM_TYPE)MQL5InfoInteger(MQL5_PROGRAM_TYPE
switch(mql_program)
  {
   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("MQL5 type value is ",mql_program);
  }
```

Información sobre el instrumento

Las funciones SymbolInfoInteger(), SymbolInfoDouble() y SymbolInfoString() sirven para obtener la información actual del mercado. Como segundo parámetro de estas funciones podemos pasar uno de los identificadores de las enumeraciones ENUM_SYMBOL_INFO_INTEGER, ENUM_SYMBOL_INFO_DOUBLE y ENUM_SYMBOL_INFO_STRING respectivamente.

Para la función SymbolInfoInteger()

ENUM_SYMBOL_INFO_INTEGER

Identificador	Descripción	Tipo de la propiedad
SYMBOL_SELECT	Indica que el símbolo ha sido seleccionado en Market Watch	bool
SYMBOL_SESSION_DEALS	Número de transacciones en la sesión actual	long
SYMBOL_SESSION_BUY_ORDERS	Número total de órdenes de compra en el momento actual	long
SYMBOL_SESSION_SELL_ORDERS	Número total de órdenes de venta en el momento actual	long
SYMBOL_VOLUME	Volume - volumen en la última transacción	long
SYMBOL_VOLUMEHIGH	Volumen máximo del día	long
SYMBOL_VOLUMELOW	Volumen mínimo del día	long
SYMBOL_TIME	Hora de la última cotización	datetime
SYMBOL_DIGITS	Número de dígitos después del punto decimal	int
SYMBOL_SPREAD	Tamaño de spread en puntos	int
SYMBOL_TICKS_BOOKDEPTH	Cantidad máxima de las solicitudes mostradas en la profundidad. Para los instrumentos sin cola de solicitudes, el valor es 0	int
SYMBOL_TRADE_CALC_MODE	Modo de calcular el coste del contrato	ENUM_SYMBOL_CALC_MODE
SYMBOL_TRADE_MODE	Tipo de ejecución de órdenes	ENUM_SYMBOL_TRADE_MODE
SYMBOL_START_TIME	Fecha de inicio de licitaciones por instrumento (normalmente se utiliza para los futuros)	datetime
SYMBOL_EXPIRATION_TIME	Fecha final de licitaciones por herramienta (normalmente se utiliza para los futuros)	datetime

SYMBOL_TRADE_STOPS_LEVEL	Margen mínimo en puntos del actual precio de cierre para la colocación de las órdenes Stop	int
SYMBOL_TRADE_FREEZE_LEVEL	Distancia de congelamiento de operaciones comerciales (en puntos)	int
SYMBOL_TRADE_EXEMODE	Modo de ejecución de transacciones	ENUM_SYMBOL_TRADE_EXECUTION
SYMBOL_SWAP_MODE	Modelo para calcular swap	ENUM_SYMBOL_SWAP_MODE
SYMBOL_SWAP_ROLLOVER3DAYS	Día de la semana para cargar la refinanciación del swap de 3 días	ENUM_DAY_OF_WEEK
SYMBOL_EXPIRATION_MODE	Banderas de los modos de expiración de la orden permitidos	int
SYMBOL_FILLING_MODE	Banderas de los modos de relleno de la orden permitidos	int
SYMBOL_ORDER_MODE	Banderas de los tipos de la orden permitidos	int

Para la función SymbolInfoDouble()

ENUM_SYMBOL_INFO_DOUBLE

Identificador	Descripción	Tipo de la propiedad
SYMBOL_BID	Bid - mejor oferta de venta	double
SYMBOL_BIDHIGH	Bid máximo del día	double
SYMBOL_BIDLOW	Bid mínimo del día	double
SYMBOL_ASK	Ask - mejor oferta de compra	double
SYMBOL_ASKHIGH	Ask máximo del día	double
SYMBOL_ASKLOW	Ask mínimo del día	double
SYMBOL_LAST	Precio de la última transacción	double
SYMBOL_LASTHIGH	Last máximo del día	double
SYMBOL_LASTLOW	Last mínimo del día	double
SYMBOL_POINT	Valor de un punto	double
SYMBOL_TRADE_TICK_VALUE	Valor SYMBOL_TRADE_TICK_VALUE_PROFIT	double
SYMBOL_TRADE_TICK_VALUE_PROFIT	Precio calculado del tick para la posición rentable	double

SYMBOL_TRADE_TICK_VALUE_LOSS	Precio calculado del tick para la posición no rentable	double
SYMBOL_TRADE_TICK_SIZE	Mínimo cambio de precio	double
SYMBOL_TRADE_CONTRACT_SIZE	Tamaño del contrato comercial	double
SYMBOL_VOLUME_MIN	Volumen mínimo para celebrar una transacción	double
SYMBOL_VOLUME_MAX	Volumen máximo para celebrar una transacción	double
SYMBOL_VOLUME_STEP	Paso mínimo del cambio de volumen para celebrar una transacción	double
SYMBOL_VOLUME_LIMIT	Volumen total máximo permitido de la posición abierta y órdenes pendientes (independientemente de la dirección) para un símbolo	double
SYMBOL_SWAP_LONG	Valor de swap long	double
SYMBOL_SWAP_SHORT	Valor de swap short	double
SYMBOL_MARGIN_INITIAL	Margen inicial (inicializador) significa el monto de fondos asegurados necesarios en la moneda de margen para abrir posiciones en el volumen de un lote. Se utiliza para verificar los fondos del cliente al entrar en el merado.	double
SYMBOL_MARGIN_MAINTENANCE	Margen de mantenimiento del instrumento. En caso de establecerlo - indica el monto del margen en la moneda de margen del instrumento que se retiene de un lote. Se utiliza para verificar los fondos del cliente cuando se cambia el estado de la cuenta del cliente. Si el margen de mantenimiento es igual a 0, se utiliza el margen inicial.	double
SYMBOL_MARGIN_LONG	Coeficiente de retención del margen para las posiciones largas	double
SYMBOL_MARGIN_SHORT	Coeficiente de retención del margen para las posiciones	double

	cortas	
SYMBOL_MARGIN_LIMIT	Coeficiente de retención del margen para las órdenes Limit	double
SYMBOL_MARGIN_STOP	Coeficiente de retención del margen para las órdenes Stop	double
SYMBOL_MARGIN_STOPLIMIT	Coeficiente de retención del margen para las órdenes Stop Limit	double
SYMBOL_SESSION_VOLUME	Volumen total de transacciones de la sesión actual	double
SYMBOL_SESSION_TURNOVER	Circulación total durante la sesión actual	double
SYMBOL_SESSION_INTEREST	Volumen total de posiciones abiertas	double
SYMBOL_SESSION_BUY_ORDERS_VOLUME	Volumen total de órdenes de compra en el momento actual	double
SYMBOL_SESSION_SELL_ORDERS_VOLUME	Volumen total de órdenes de venta en el momento actual	double
SYMBOL_SESSION_OPEN	Precio de apertura de la sesión	double
SYMBOL_SESSION_CLOSE	Precio de cierre de la sesión	double
SYMBOL_SESSION_AW	Precio medio ponderado de la sesión	double
SYMBOL_SESSION_PRICE_SETTLEMENT	Precio de entrega para la sesión actual	double
SYMBOL_SESSION_PRICE_LIMIT_MIN	Precio mínimo aceptable para la sesión	double
SYMBOL_SESSION_PRICE_LIMIT_MAX	Precio máximo aceptable para la sesión	double

Para la función SymbolInfoString()

ENUM_SYMBOL_INFO_STRING

Identificador	Descripción	Tipo de la propiedad
SYMBOL_CURRENCY_BASE	Divisa básica del instrumento	string
SYMBOL_CURRENCY_PROFIT	Divisa de beneficio	string
SYMBOL_CURRENCY_MARGIN	Divisa de margen	string
SYMBOL_BANK	Fuente de cotización actual	string
SYMBOL_DESCRIPTION	Descripción literal del símbolo	string

SYMBOL_ISIN	Nombre del símbolo comercial en el sistema de los códigos internacionales para identificación de valores — ISIN (International Securities Identification Number). El código internacional de identificación de un valor es un código de 12 dígitos que contiene letras y cifras y que identifica de forma unívoca a un valor mobiliario a nivel internacional. La presencia de esta propiedad del símbolo se determina en el lado del servidor comercial.	string
SYMBOL_PATH	Ruta en el árbol de símbolos	string

Para cada símbolo se puede indicar varios modos de plazo de vigencia (vencimiento) de las órdenes pendientes. Cada modo tiene su bandera; las banderas pueden ser combinadas mediante la operación del lógico **OR** (|), por ejemplo, SYMBOL_EXPIRATION_GTC|SYMBOL_EXPIRATION_SPECIFIED. Para comprobar si un modo en concreto está permitido para un instrumento, hay que comparar el resultado del lógico **AND** (&) con la bandera del modo.

Si para el símbolo se especifica la bandera SYMBOL_EXPIRATION_SPECIFIED, entonces cuando se envía la orden pendiente, se puede indicar con precisión hasta que momento dicha orden está vigente.

Identificador	Valor	Descripción
SYMBOL_EXPIRATION_GTC	1	La orden está vigente sin restricción de tiempo hasta su explícita cancelación
SYMBOL_EXPIRATION_DAY	2	La orden está vigente hasta que se termine el día
SYMBOL_EXPIRATION_SPECIFIED	4	El plazo de vencimiento se indica en la orden
SYMBOL_EXPIRATION_SPECIFIED_DAY	8	El día de vencimiento se especifica en la orden

Ejemplo:

```
//+------------------------------------------------------------------+
//| comprueba si el modo especificado de vencimiento está permitido|
//+------------------------------------------------------------------+
bool IsExpirationTypeAllowed(string symbol,int exp_type)
  {
//--- obtenemos el valor de la propiedad que describe los modos de vencimiento permit
```

```
   int expiration=(int)SymbolInfoInteger(symbol,SYMBOL_EXPIRATION_MODE);
//--- devolvemos true, si el modo exp_type está permitido
   return((expiration & exp_type)==exp_type);
  }
```

Cuando enviamos una orden, podemos indicar la política de relleno del volumen solicitado en esa orden. Las opciones de ejecución de la orden que están permitidas, respecto al volumen para cada símbolo, se especifican en la tabla de abajo. Para cada símbolo se puede indicar no sólo un modo, sino varios, usando las banderas de combinación. Las banderas pueden ser combinadas a través de la operación del lógico **OR** (|) por ejemplo, SYMBOL_FILLING_ALL_OR_NONE| SYMBOL_CANCEL_REMAIND. Para comprobar si un modo en concreto está permitido para un instrumento, hay que comparar el resultado del lógico **AND** (&) con la bandera del modo.

Identificador	Valor	Descripción
SYMBOL_FILLING_ALL_OR_NONE	1	Se especifica "Todo o nada". Si el volumen especificado en la orden con el precio indicado no llega a llenarse, entonces la orden se cancela y la transacción no se realiza
SYMBOL_CANCEL_REMAIND	2	Si con el precio indicado en la orden sólo una parte del volumen puede llenarse, la transacción se lleva a cabo según el volumen alcanzado. El resto de la orden se quita y no se presenta ninguna orden nueva
SYMBOL_RETURN_REMAIND	4	La transacción se realiza según el precio especificado en la orden dentro del margen del volumen disponible. Para el resto se coloca una orden nueva con el precio de antes

Ejemplo:

```
//+------------------------------------------------------------------+
//| comprueba si el modo especificado de vencimiento está permitido |
//+------------------------------------------------------------------+
bool IsFillingTypeAllowed(string symbol,int fill_type)
  {
//--- obtenemos el valor de la propiedad que describe el modo de relleno
   int filling=(int)SymbolInfoInteger(symbol,SYMBOL_FILLING_MODE);
//--- devolvemos true, si el modo fill_type está permitido
   return((filling & fill_type)==fill_type);
  }
```

Cuando se envía una solicitud comercial usando la función OrderSend(), para algunas operaciones hace falta especificar el tipo de la oreden desde la enumeración ENUM_ORDER_TYPE. No todos los tipos de órdenes pueden estar permitidos para un símbolo específico, la propiedad SYMBOL_ORDER_MODE describe las banderas de los tipos permitidos.

Identificador	Valor	Descripción
SYMBOL_ORDER_MARKET	1	Están permitidas las órdenes de mercado (Buy y Sell en el mercado sin especificar el precio de transacción)
SYMBOL_ORDER_LIMIT	2	Están permitidas las órdenes limitadas (Buy Limit y Sell Limit)
SYMBOL_ORDER_STOP	4	Están permitidas las órdenes Stop (Buy Stop y Sell Stop)
SYMBOL_ORDER_STOP_LIMIT	8	Están permitidas las órdenes Stop Limit (Buy Stop Limit y Sell Stop Limit)
SYMBOL_ORDER_SL	16	Está permitida la colocación de Stop Loss
SYMBOL_ORDER_TP	32	Está permitida la colocación de Take Profit

Ejemplo:

```
//+------------------------------------------------------------------+
//| La función imprime los tipos de órdenes permitidos para el símbolo   |
//+------------------------------------------------------------------+
void Check_SYMBOL_ORDER_MODE(string symbol)
  {
//--- obtenemos el valor de la propiedad que describe los tipos de órdenes permitidos
   int symbol_order_mode=(int)SymbolInfoInteger(symbol,SYMBOL_ORDER_MODE);
//--- chequeo para órdenes de mercado (Market Execution)
   if((SYMBOL_ORDER_MARKET&symbol_order_mode)==SYMBOL_ORDER_MARKET)
      Print(symbol+": Las órdenes de mercado están permitidas (no se requiere indicar
//--- chequeo para órdenes del tipo Limit
   if((SYMBOL_ORDER_LIMIT&symbol_order_mode)==SYMBOL_ORDER_LIMIT)
      Print(symbol+": Las órdenes Buy Limit y Sell Limit están permitidas");
//--- chequeo para órdenes del tipo Stop
   if((SYMBOL_ORDER_STOP&symbol_order_mode)==SYMBOL_ORDER_STOP)
      Print(symbol+": Las órdenes Buy Stop y Sell Stop están permitidas");
//--- chequeo para órdenes del tipo Stop Limit
   if((SYMBOL_ORDER_STOP_LIMIT&symbol_order_mode)==SYMBOL_ORDER_STOP_LIMIT)
      Print(symbol+": Las órdenes Buy Stop Limit y Sell Stop Limit están permitidas")
//--- comprobación de posibilidad de colocación de las órdenes Stop Loss
   if((SYMBOL_ORDER_SL&symbol_order_mode)==SYMBOL_ORDER_SL)
      Print(symbol+": Las órdenes Stop Loss están permitidas");
//--- comprobación de posibilidad de colocación de las órdenes Take Profit
   if((SYMBOL_ORDER_TP&symbol_order_mode)==SYMBOL_ORDER_TP)
      Print(symbol+": Las órdenes Take Profit están permitidas");
//---
  }
```

La enumeración ENUM_SYMBOL_CALC_MODE sirve para obtener la información sobre el cálculo del monto de los fondos prendarios (monto de los requerimientos de margen).

ENUM_SYMBOL_CALC_MODE

Identificador	Descripción	Formula
SYMBOL_CALC_MODE_FOREX	Forex mode - cálculo de beneficio y margen para Forex	Margin: Lots*Contract_Size/ Leverage Profit: (close_price-open_price) *Contract_Size*Lots
SYMBOL_CALC_MODE_FUTURES	Futures mode - cálculo de beneficio y margen para futuros	Margin: Lots *ContractSize*MarketPrice*Percentage/100 Profit: (close_price-open_price) *Contract_Size*Lots
SYMBOL_CALC_MODE_CFD	CFD mode - cálculo de beneficio y margen para CFD	
SYMBOL_CALC_MODE_CFDINDEX	CFD index mode - cálculo de beneficio y margen para CFD por índices	Margin: (Lots*ContractSize*MarketPrice)*TickPrice/TickSize Profit: (close_price-open_price)

		*Contract_Size*Lots
SYMBOL_CALC_MODE_CFDLEVERAGE	CFD Leverage mode – cálculo de beneficio y margen para CFD en caso del trading con apalancamiento financiero	Margin: (Lots*ContractSize*MarketPrice*Percentage)/Leverage Profit: (close_price-open_price) *Contract_Size*Lots

Existen varias formas de tradear con el instrumento. La información sobre los regímenes de comerciar con cada instrumento en concreto se especifica en los valores de la enumeración ENUM_SYMBOL_TRADE_MODE.

ENUM_SYMBOL_TRADE_MODE

Identificador	Descripción
SYMBOL_TRADE_MODE_DISABLED	Trading por el símbolo prohibido
SYMBOL_TRADE_MODE_LONGONLY	Sólo compras
SYMBOL_TRADE_MODE_SHORTONLY	Sólo ventas
SYMBOL_TRADE_MODE_CLOSEONLY	Permitidas sólo operaciones de cierre de posiciones
SYMBOL_TRADE_MODE_FULL	Sin restricciones de las operaciones comerciales

En la enumeración ENUM_SYMBOL_TRADE_EXECUTION se especifican los posibles regímenes de llevar a cabo las transacciones por cada instrumento en concreto.

ENUM_SYMBOL_TRADE_EXECUTION

Identificador	Descripción
SYMBOL_TRADE_EXECUTION_REQUEST	Ejecución por Pedido
SYMBOL_TRADE_EXECUTION_INSTANT	Ejecución Instantánea
SYMBOL_TRADE_EXECUTION_MARKET	Ejecución de órdenes por Mercado
SYMBOL_TRADE_EXECUTION_EXCHANGE	Ejecución por Bolsa

Los modos de calcular los swaps al cambiar de posición se especifican el la enumeración ENUM_SYMBOL_SWAP_MODE. El modo de calcular los swaps determina las unidades de medición de lsos parámetros SYMBOL_SWAP_LONG y SYMBOL_SWAP_SHORT. Por ejemplo, si los swaps se calculan en la divisa del depósito del cliente, en los parámetros el volumen de los swaps calculados se indica precisamente en la divisa del depósito del cliente.

ENUM_SYMBOL_SWAP_MODE

Identificador	Descripción
SYMBOL_SWAP_MODE_DISABLED	No hay swaps
SYMBOL_SWAP_MODE_POINTS	Swaps se calculan en puntos
SYMBOL_SWAP_MODE_CURRENCY_SYMBOL	Swaps se calculan en dinero en divisa base del símbolo
SYMBOL_SWAP_MODE_CURRENCY_MARGIN	Swaps se calculan en dinero en divisa marginal del símbolo
SYMBOL_SWAP_MODE_CURRENCY_DEPOSIT	Swaps se calculan en dinero en divisa del depósito del cliente
SYMBOL_SWAP_MODE_INTEREST_CURRENT	Swaps se calculan en por cientos anuales del precio del instrumento para el momento de cálculo del swap (régimen bancario es de 360 días al año)
SYMBOL_SWAP_MODE_INTEREST_OPEN	Swaps se calculan en por cientos anuales del precio de apertura de la posición para el símbolo (régimen bancario es de 360 días al año)
SYMBOL_SWAP_MODE_REOPEN_CURRENT	Swaps se calculan por reapertura de posiciones. Al final de la jornada de trading la posición se cierra forzosamente. Al día siguiente la posición se vuelve a abrir por el precio de cierre +/- el número de puntos especificado (en los parámetros SYMBOL_SWAP_LONG y SYMBOL_SWAP_SHORT)
SYMBOL_SWAP_MODE_REOPEN_BID	Swaps se calculan por reapertura de posiciones. Al final de la jornada de trading la posición se cierra forzosamente. Al día siguiente la posición se vuelve a abrir por el precio actual Bid +/- el número de puntos especificado (en los parámetros SYMBOL_SWAP_LONG y SYMBOL_SWAP_SHORT)

La enumeración ENUM_DAY_OF_WEEK sirve para indicar el día de la semana.

ENUM_DAY_OF_WEEK

Identificador	Descripción
SUNDAY	Domingo
MONDAY	Lunes
TUESDAY	Martes
WEDNESDAY	Miércoles

THURSDAY	Jueves
FRIDAY	Viernes
SATURDAY	Sábado

Información sobre la cuenta

Las funciones AccountInfoInteger(), AccountInfoDouble() y AccountInfoString() sirven para obtener la información sobre la cuenta corriente. Los valores de parámetros de estas funciones aceptan los valores de las correspondientes enumeraciones ENUM_ACCOUNT_INFO.

Para la función AccountInfoInteger()

ENUM_ACCOUNT_INFO_INTEGER

Identificador	Descripción	Tipo de la propiedad
ACCOUNT_LOGIN	Número de cuenta	long
ACCOUNT_TRADE_MODE	Account trade mode	ENUM_ACCOUNT_TRADE_MODE
ACCOUNT_LEVERAGE	Cuantía de apalancamiento concedido	long
ACCOUNT_LIMIT_ORDERS	El número máximo permitido de las órdenes pendientes activas	int
ACCOUNT_MARGIN_SO_MODE	Modo de establecimiento del nivel mínimo permitido de la margen	ENUM_ACCOUNT_STOPOUT_MODE
ACCOUNT_TRADE_ALLOWED	Permiso del trading para la cuenta corriente	bool
ACCOUNT_TRADE_EXPERT	Permiso del trading para un Asesor Experto	bool

Para la función AccountInfoDouble()

ENUM_ACCOUNT_INFO_DOUBLE

Identificador	Descripción	Tipo de la propiedad
ACCOUNT_BALANCE	Balance de cuenta en divisa del depósito	double
ACCOUNT_CREDIT	Cuantía de crédito concedido en divisa del depósito	double
ACCOUNT_PROFIT	Cuantía del beneficio actual en la cuenta en divisa del depósito	double
ACCOUNT_EQUITY	Cuantía de fondos propios en la cuenta en divisa del depósito	double
ACCOUNT_MARGIN	Cuantía de margen reservada en la cuenta en divisa del depósito	double

ACCOUNT_FREEMARGIN	Cuantía de fondos disponibles en la cuenta en divisa del depósito para la apertura de una posición	double
ACCOUNT_MARGIN_LEVEL	Nivel de margen en la cuenta en por cientos	double
ACCOUNT_MARGIN_SO_CALL	Nivel de margen que requiere el recargo de la cuenta. Dependiendo del ACCOUNT_MARGIN_SO_MODE establecido, éste va en por cientos o en divisa del depósito	double
ACCOUNT_MARGIN_SO_SO	Nivel de margen; al llegar a este nivel, la posición menos rentable se cierra automáticamente. Dependiendo del ACCOUNT_MARGIN_SO_MODE establecido, éste va en por cientos o en divisa del depósito	double

Para la función AccountInfoString()

ENUM_ACCOUNT_INFO_STRING

Identificador	Descripción	Tipo de la propiedad
ACCOUNT_NAME	Nombre de cliente	string
ACCOUNT_SERVER	Nombre de servidor comercial	string
ACCOUNT_CURRENCY	Divisa de depósito	string
ACCOUNT_COMPANY	Nombre de la empresa que atiende la cuenta	string

Hay varios tipos de cuentas que pueden estar abiertos en el servidor comercial. Para averiguar en qué tipo de cuenta opera un programa MQL5, se utiliza la enumeración ENUM_ACCOUNT_TRADE_MODE.

ENUM_ACCOUNT_TRADE_MODE

Identificador	Descripción
ACCOUNT_TRADE_MODE_DEMO	Cuenta comercial de demostración (Demo)
ACCOUNT_TRADE_MODE_CONTEST	Cuenta comercial de concurso (Contest)
ACCOUNT_TRADE_MODE_REAL	Cuenta comercial real (Real)

La situación del cierre forzoso Stop Out surge cuando faltan propios fondos para mantener las posiciones abiertas. El nivel mínimo de la margen que provoca Stop Out se puede establecer en por

cientos o en dinero. La enumeración ENUM_ACCOUNT_STOPOUT_MODE sirve para averiguar el modo establecido para una cuenta en cuestión.

ENUM_ACCOUNT_STOPOUT_MODE

Identificador	Descripción
ACCOUNT_STOPOUT_MODE_PERCENT	Nivel en por cientos
ACCOUNT_STOPOUT_MODE_MONEY	Nivel en dinero

Un ejemplo del script que muestra la información breve sobre la cuenta.

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- nombre de la empresa
   string company=AccountInfoString(ACCOUNT_COMPANY);
//--- nombre del cliente
   string name=AccountInfoString(ACCOUNT_NAME);
//--- número de la cuenta
   long login=AccountInfoInteger(ACCOUNT_LOGIN);
//--- nombre del servidor
   string server=AccountInfoString(ACCOUNT_SERVER);
//--- divisa de la cuenta
   string currency=AccountInfoString(ACCOUNT_CURRENCY);
//--- cuenta de demostración, de concurso o real
   ENUM_ACCOUNT_TRADE_MODE account_type=(ENUM_ACCOUNT_TRADE_MODE)AccountInfoInteger(A
//--- ahora transformaremos el valor de la enumeración en una forma comprensible
   string trade_mode;
   switch(account_type)
     {
      case  ACCOUNT_TRADE_MODE_DEMO:
         trade_mode="demo";
         break;
      case  ACCOUNT_TRADE_MODE_CONTEST:
         trade_mode="de concurso";
         break;
      default:
         trade_mode="real";
         break;
     }
//--- Stop Out se establece en por cientos o dinero
   ENUM_ACCOUNT_STOPOUT_MODE stop_out_mode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoInte
//--- obtenemos valores de niveles cuando ocurren Margin Call y Stop Out
   double margin_call=AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL);
   double stop_out=AccountInfoDouble(ACCOUNT_MARGIN_SO_SO);
//--- visualizaremos la información breve sobre la cuenta
```

```
   PrintFormat("La cuenta del cliente '%s' #%d %s está abierta en '%s' en el servidor
               name,login,trade_mode,company,server);
   PrintFormat("Moneda de la cuenta - %s, el nivel MarginCall y StopOut se establece
               currency,(stop_out_mode==ACCOUNT_STOPOUT_MODE_PERCENT)?"en por cientos
   PrintFormat("Nivel MarginCall=%G, nivel StopOut=%G",margin_call,stop_out);
  }
```

Estadística de simulación

Una vez finalizado el proceso de simulación, se calculan los indicadores estadísticos de los resultados comerciales según diferentes parámetros. Los valores de estos indicadores se puede conseguir a través de la función TesterStatistics(), especificando previamente el identificador del indicador desde de la enumeración ENUM_STATISTICS.

Aunque para calcular los datos estadísticos se utilizan dos tipos de parámetros - int y double - la función devuelve todos los valores como double. Por defecto, todos los valores estadísticos del tipo double se expresan en la moneda del depósito, si no está especificado lo otro.

ENUM_STATISTICS

Identificador	Descripción del parámetro estadístico	Tipo
STAT_INITIAL_DEPOSIT	Valor del depósito inicial	double
STAT_WITHDRAWAL	Fondos retirados de la cuenta	double
STAT_PROFIT	Beneficio neto tras la simulación, suma de STAT_GROSS_PROFIT y STAT_GROSS_LOSS (STAT_GROSS_LOSS siempre es menos o igual a cero)	double
STAT_GROSS_PROFIT	Beneficio bruto, importe de todas las transacciones rentables (con resultados positivos). Su valor es superior o igual a cero	double
STAT_GROSS_LOSS	Pérdidas brutas, importe de todas las transacciones de pérdidas (con resultados negativos). Su valor es inferior o igual a cero	double
STAT_MAX_PROFITTRADE	Beneficio máximo – el valor máximo entre todas las transacciones rentables. Su valor es superior o igual a cero	double
STAT_MAX_LOSSTRADE	Pérdida máxima – el valor mínimo entre todas las transacciones de pérdidas. Su valor es inferior o igual a cero	double
STAT_CONPROFITMAX	Beneficio máximo en una secuencia de transacciones rentables. Su valor es superior o igual a cero	double
STAT_CONPROFITMAX_TRADES	Número de transacciones que	int

	han formado STAT_CONPROFITMAX (beneficio máximo en una secuencia de transacciones rentables)	
STAT_MAX_CONWINS	Beneficio total en la serie más larga de transacciones rentables	double
STAT_MAX_CONPROFIT_TRADES	Numero de transacciones en la serie más larga de transacciones rentables STAT_MAX_CONWINS	int
STAT_CONLOSSMAX	Pérdida máxima en una secuencia de transacciones de pérdidas. Su valor es inferior o igual a cero	double
STAT_CONLOSSMAX_TRADES	Número de transacciones que han formado STAT_CONLOSSMAX (pérdida máxima en una secuencia de transacciones de pérdidas)	int
STAT_MAX_CONLOSSES	Pérdidas totales en la serie más larga de transacciones de pérdidas	double
STAT_MAX_CONLOSS_TRADES	Número de transacciones en la serie más larga de transacciones de pérdidas STAT_MAX_CONLOSSES	int
STAT_BALANCEMIN	Valor mínimo del balance	double
STAT_BALANCE_DD	Reducción máxima del balance en dinero. En el proceso de trading el balance puede sufrir varias reducciones, se coge el valor máximo.	double
STAT_BALANCEDD_PERCENT	Reducción del balance en por cientos que ha sido detectada en el momento de la reducción máxima del balance en dinero (STAT_BALANCE_DD).	double
STAT_BALANCE_DDREL_PERCENT	Reducción máxima del balance en por cientos. En el proceso de trading el balance puede sufrir varias reducciones, para cada una de ellas se calcula el valor relativo de reducción en	double

	por cientos. Se devuelve el valor máximo	
STAT_BALANCE_DD_RELATIVE	Reducción del balance en dinero que ha sido detectada en el momento de la reducción máxima del balance en por cientos (STAT_BALANCE_DDREL_PERCENT).	double
STAT_EQUITYMIN	Valor mínimo de equidad	double
STAT_EQUITY_DD	Reducción máxima de equidad en dinero. En el proceso de trading la equidad puede sufrir varias reducciones, se coge el valor máximo.	double
STAT_EQUITYDD_PERCENT	Reducción de equidad en por cientos que ha sido detectada en el momento de la reducción máxima de equidad en dinero (STAT_EQUITY_DD).	double
STAT_EQUITY_DDREL_PERCENT	Reducción máxima de equidad en por cientos. En el proceso de trading la equidad puede sufrir varias reducciones, para cada una de ellas se calcula el valor relativo de reducción en por cientos. Se devuelve el valor máximo	double
STAT_EQUITY_DD_RELATIVE	Reducción de equidad en dinero que ha sido detectada en el momento de la reducción máxima de equidad en por cientos (STAT_EQUITY_DDREL_PERCENT).	double
STAT_EXPECTED_PAYOFF	Beneficio esperado	double
STAT_PROFIT_FACTOR	Factor de beneficio - relación entre STAT_GROSS_PROFIT/ STAT_GROSS_LOSS. Si STAT_GROSS_LOSS=0, entonces el Factor de Beneficio obtiene el valor DBL_MAX	double
STAT_RECOVERY_FACTOR	Factor de recuperación - relación entre STAT_PROFIT/ STAT_BALANCE_DD	double

STAT_SHARPE_RATIO	Ratio de Sharpe	double
STAT_MIN_MARGINLEVEL	Valor mínimo alcanzado del nivel de margen	double
STAT_CUSTOM_ONTESTER	Valor del criterio de optimización de usuario, calculado y devuelto por la función OnTester()	double
STAT_DEALS	Número de transacciones realizadas	int
STAT_TRADES	Número de trades	int
STAT_PROFIT_TRADES	Trades rentables	int
STAT_LOSS_TRADES	Trades irrentables	int
STAT_SHORT_TRADES	Trades cortos	int
STAT_LONG_TRADES	Trades largos	int
STAT_PROFIT_SHORTTRADES	Trades cortos rentables	int
STAT_PROFIT_LONGTRADES	Trades largos rentables	int
STAT_PROFITTRADES_AVGCON	Longitud media de una serie rentable de trades	int
STAT_LOSSTRADES_AVGCON	Longitud media de una serie irrentable de trades	int

Constantes comerciales

Varias constantes que se usan para programar las estrategias comerciales están divididas en los siguientes grupos:

- Información sobre datos históricos del instrumento – la obtención de la información general sobre el instrumento financiera;
- Propiedades de órdenes – obtención de información sobre las órdenes comerciales;
- Propiedades de posiciones – obtención de información sobre las posiciones actuales;
- Propiedades de transacciones – obtención de información sobre las transacciones realizadas;
- Tipos de operaciones comerciales – descripción de las operaciones comerciales disponibles;
- Tipos de transacciones comerciales - descripción de posibles tipos de transacciones comerciales;
- Tipos de órdenes en profundidad de mercado - separación de las órdenes según la dirección de operación comercial solicitada.

Información sobre datos históricos del instrumento

Cuando se accede a las series temporales se usa la función SeriesInfoInteger() para obtener la información adicional sobre el instrumento. El identificador de la propiedad requerida se pasa como el parámetro de esta función. Este identificador puede adquirir uno de los valores de la enumeración ENUM_SERIES_INFO_INTEGER.

ENUM_SERIES_INFO_INTEGER

Identificador	Descripción	Tipo de la propiedad
SERIES_BARS_COUNT	Número de barras para el símbolo-período en el momento actual	long
SERIES_FIRSTDATE	La primera fecha para el símbolo-período en el momento actual	datetime
SERIES_LASTBAR_DATE	Información sobre datos históricos del instrumento	datetime
SERIES_SERVER_FIRSTDATE	La primera fecha en el historial para el símbolo en el servidor independientemente del período	datetime
SERIES_TERMINAL_FIRSTDATE	La primera fecha en el historial para el símbolo en el terminal de cliente independientemente del período	datetime
SERIES_SYNCHRONIZED	Significa la sincronización de datos para el símbolo/período en este momento	bool

Propiedades de órdenes

Ordenaciones de ejecutar las operaciones comerciales se formalizan mediante las órdenes. Cada orden posee una multitud de propiedades para la lectura. La información acerca de ellas se obtiene a través de la función OrderGet...() y HistoryOrderGet...().

Para las funciones OrderGetInteger() y HistoryOrderGetInteger()

ENUM_ORDER_PROPERTY_INTEGER

Identificador	Descripción	Tipo
ORDER_TIME_SETUP	Hora de establecimiento de la orden	datetime
ORDER_TYPE	Tipo de la orden	ENUM_ORDER_TYPE
ORDER_STATE	Estatus de la orden	ENUM_ORDER_STATE
ORDER_TIME_EXPIRATION	Plazo de expiración de la orden	datetime
ORDER_TIME_DONE	Hora de ejecución o cancelación de la orden	datetime
ORDER_TIME_SETUP_MSC	Tiempo de colocación de la orden para la ejecución en milisegundos desde 01.01.1970	long
ORDER_TIME_DONE_MSC	Tiempo de ejecución / retirada de la orden en milisegundos desde 01.01.1970	long
ORDER_TYPE_FILLING	Tipo de ejecución según el resto	ENUM_ORDER_TYPE_FILLING
ORDER_TYPE_TIME	Tiempo de vida de la orden	ENUM_ORDER_TYPE_TIME
ORDER_MAGIC	Identificador del Asesor Experto que ha colocado la orden (sirve para que cada Asesor Experto ponga su único número personal)	long
ORDER_POSITION_ID	Identificador de posición que se coloca en la orden a la hora de su ejecución. Cada orden ejecutada provoca una transacción que abre una nueva posición, o cambia una que ya existe. El identificador de esta misma posición se establece para la orden ejecutada en este momento.	long

Para las funciones OrderGetDouble() y HistoryOrderGetDouble()

ENUM_ORDER_PROPERTY_DOUBLE

Identificador	Descripción	Tipo
ORDER_VOLUME_INITIAL	Volumen inicial de la orden	double
ORDER_VOLUME_CURRENT	Volumen actual de la orden	double
ORDER_PRICE_OPEN	Precio especificado en la orden	double
ORDER_SL	Nivel Stop Loss	double
ORDER_TP	Nivel Take Profit	double
ORDER_PRICE_CURRENT	Precio actual del símbolo de la orden	double
ORDER_PRICE_STOPLIMIT	Precio Limit de la orden al activarse StopLimit	double

Para las funciones OrderGetString() y HistoryOrderGetString()

ENUM_ORDER_PROPERTY_STRING

Identificador	Descripción	Tipo
ORDER_SYMBOL	Símbolo de la orden	string
ORDER_COMMENT	Comentario	string

Cuando se envía una solicitud comercial mediante la función OrderSend(), algunas operaciones requieren la indicación del tipo de la orden. El tipo de la orden se introduce en el campo *type* de una estructura especial MqlTradeRequest y puede adquirir los valores de la enumeración ENUM_ORDER_TYPE.

ENUM_ORDER_TYPE

Identificador	Descripción
ORDER_TYPE_BUY	Orden de mercado para la compra
ORDER_TYPE_SELL	Orden de mercado para la venta
ORDER_TYPE_BUY_LIMIT	Orden pendiente Buy Limit
ORDER_TYPE_SELL_LIMIT	Orden pendiente Sell Limit
ORDER_TYPE_BUY_STOP	Orden pendiente Buy Stop
ORDER_TYPE_SELL_STOP	Orden pendiente Sell Stop
ORDER_TYPE_BUY_STOP_LIMIT	Al alcanzar el precio de la orden se coloca la orden pendiente Buy Limit por el precio

	StopLimit
ORDER_TYPE_SELL_STOP_LIMIT	Al alcanzar el precio de la orden se coloca la orden pendiente Sell Limit por el precio StopLimit

Cada orden tiene su estatus que describe su estado. Para obtener más detalles utilice la función OrderGetInteger() o HistoryOrderGetInteger() con el modificador ORDER_STATE. Los valores admisibles se almacenan en la enumeración ENUM_ORDER_STATE.

ENUM_ORDER_STATE

Identificador	Descripción
ORDER_STATE_STARTED	Orden verificada pero aún sin aceptar por el corredor
ORDER_STATE_PLACED	Orden aceptada
ORDER_STATE_CANCELED	Orden retirada por el cliente
ORDER_STATE_PARTIAL	Orden ejecutada parcialmente
ORDER_STATE_FILLED	Orden ejecutada totalmente
ORDER_STATE_REJECTED	Orden rechazada
ORDER_STATE_EXPIRED	Orden retirada por expirarse el plazo
ORDER_STATE_REQUEST_ADD	Orden en el estado de registro (colocación en el sistema de trading)
ORDER_STATE_REQUEST_MODIFY	Orden en el estado de modificación (cambio de parámetros)
ORDER_STATE_REQUEST_CANCEL	Orden en el estado de eliminación (eliminación del sistema de trading)

Cuando se envía una solicitud comercial mediante la función OrderSend() se puede definir la política de ejecución de la orden en el campo *type_filling* en una estructura especial MqlTradeRequest; se admiten los valores de la enumeración ENUM_ORDER_TYPE_FILLING. Para obtener el valor de esta propiedad utilice la función OrderGetInteger() o HistoryOrderGetInteger() con el modificador ORDER_TYPE_FILLING.

ENUM_ORDER_TYPE_FILLING

Identificador	Descripción
ORDER_FILLING_FOK	Esta política de ejecución significa que la orden puede ser ejecutada exclusivamente en el volumen especificado. Si en este momento en el mercado no hay volumen necesario del

	instrumento financiero requerido, esta orden no se ejecutará. El volumen necesario puede ser cubierto por varias ofertas disponibles en este momento en el mercado.
ORDER_FILLING_IOC	Significa que el trader acepta realizar la transacción en el volumen máximo disponible en el mercado dentro del margen especificado en la orden. Si la ejecución completa no es posible, la orden será ejecutada en el volumen disponible, y el resto del volumen no cubierto será cancelado.
ORDER_FILLING_RETURN	Esta política de ejecución se utiliza para las órdenes de mercado (ORDER_TYPE_BUY y ORDER_TYPE_SELL), limitadas y limitadas con Stop (ORDER_TYPE_BUY_LIMIT, ORDER_TYPE_SELL_LIMIT, ORDER_TYPE_BUY_STOP_LIMIT y ORDER_TYPE_SELL_STOP_LIMIT), y sólo en los modos "Ejecución por Mercado" y "Ejecución por Bolsa". En caso de la ejecución parcial, una orden limitada o de mercado con el volumen no cubierto no se anula, sino sigue estando vigente. Para las órdenes ORDER_TYPE_BUY_STOP_LIMIT y ORDER_TYPE_SELL_STOP_LIMIT en el momento de la activación será creada la orden correspondiente limitada ORDER_TYPE_BUY_LIMIT/ ORDER_TYPE_SELL_LIMIT con el tipo de ejecución ORDER_FILLING_RETURN.

Se puede determinar el plazo de vigencia de la orden en el campo *type_time* de la estructura especial MqlTradeRequest a la hora de enviar la solicitud comercial mediante OrderSend(). Se admiten los valores de la enumeración ENUM_ORDER_TYPE_TIME. Para obtener el valor de esta propiedad utilice la función OrderGetInteger() o HistoryOrderGetInteger() con el modificador ORDER_TYPE_TIME.

ENUM_ORDER_TYPE_TIME

Identificador	Descripción
ORDER_TIME_GTC	Orden estará en la cola hasta que se retire
ORDER_TIME_DAY	Orden estará vigente sólo en el transcurso del día comercial corriente
ORDER_TIME_SPECIFIED	Orden estará vigente hasta que se expire el plazo de vigencia
ORDER_TIME_SPECIFIED_DAY	Orden se mantiene activa hasta las 00:00 del

	día especificado. Si está hora no cae en ninguna sesión de trading, la expiración llega a la hora de trading más cercana.

Propiedades de posiciones

El resultado de ejecución de las operaciones comerciales es la apertura de una posición, cambio de su volumen y/o dirección, o su desaparición. Las operaciones comerciales se llevan a cabo a base de las órdenes enviadas por la función OrderSend() en forma de las solicitudes comerciales. Para cada instrumento financiero (símbolo) se prevé posible sólo una posición abierta. Una posición tiene un conjunto de propiedades disponibles para la lectura para las funciones PositionGet...().

Para la función PositionGetInteger()

ENUM_POSITION_PROPERTY_INTEGER

Identificador	Descripción	Tipo
POSITION_TIME	Hora de apertura de posición	datetime
POSITION_TIME_MSC	Tiempo de apertura de la posición en milisegundos desde 01.01.1970	long
POSITION_TIME_UPDATE	Tiempo de modificación de la posición en segundos desde 01.01.1970	long
POSITION_TIME_UPDATE_MSC	Tiempo de modificación de la posición en milisegundos desde 01.01.1970	long
POSITION_TYPE	Tipo de posición	ENUM_POSITION_TYPE
POSITION_MAGIC	Magic number para la posición (véase ORDER_MAGIC)	long
POSITION_IDENTIFIER	Identificador de posición es un número único que se adjudica a cada una de las posiciones abiertas y no se cambia a lo largo de su existencia. La rotación de posición no cambia su identificador.	long

Para la función PositionGetDouble()

ENUM_POSITION_PROPERTY_DOUBLE

Identificador	Descripción	Tipo
POSITION_VOLUME	Volumen de posición	double
POSITION_PRICE_OPEN	Precio de posición	double
POSITION_SL	Nivel Stop Loss para una posición abierta	double
POSITION_TP	Nivel Take Profit para una posición abierta	double

POSITION_PRICE_CURRENT	Precio actual para el símbolo	double
POSITION_COMMISSION	Comisión	double
POSITION_SWAP	Swap acumulado	double
POSITION_PROFIT	Beneficio corriente	double

Para la función PositionGetString()

ENUM_POSITION_PROPERTY_STRING

Identificador	Descripción	Tipo
POSITION_SYMBOL	Símbolo de posición abierta	string
POSITION_COMMENT	Comentarios de posición	string

La dirección de una posición abierta (compra o venta) se determina con los valores de la enumeración ENUM_POSITION_TYPE. Para obtener el tipo de una posición abierta, utilice la función PositionGetInteger() con el modificador POSITION_TYPE.

ENUM_POSITION_TYPE

Identificador	Descripción
POSITION_TYPE_BUY	Compra
POSITION_TYPE_SELL	Venta

Propiedades de transacciones

Una transacción refleja el hecho de ejecución de una operación comercial a base de una orden que contiene una disposición comercial. Cada transacción se describe por las propiedades que permiten obtener información sobre ella. Para leer los valores de las propiedades se utilizan las funciones del tipo HistoryDealGet...() que devuelven los valores de las enumeraciones correspondientes.

Para la función HistoryDealGetInteger()

ENUM_DEAL_PROPERTY_INTEGER

Identificador	Descripción	Tipo
DEAL_ORDER	Orden a base de la cual la transacción ha sido ejecutada	long
DEAL_TIME	Hora de ejecución de transacción	datetime
DEAL_TIME_MSC	Tiempo de ejecución de la transacción en milisegundos desde 01.01.1970	long
DEAL_TYPE	Tipo de transacción	ENUM_DEAL_TYPE
DEAL_ENTRY	Dirección de transacción - entra en el mercado, sale del mercado o da la vuelta	ENUM_DEAL_ENTRY
DEAL_MAGIC	Magic number para la transacción (véase ORDER_MAGIC)	long
DEAL_POSITION_ID	Identificador de posición, en la apertura, modificación o cierre de la cual participa esta transacción. Cada posición tiene su único identificador que se asigna a todas las transacciones realizadas con el instrumento durante toda la vida de la posición.	long

Para la función HistoryDealGetDouble()

ENUM_DEAL_PROPERTY_DOUBLE

Identificador	Descripción	Tipo
DEAL_VOLUME	Volumen de transacción	double
DEAL_PRICE	Precio de transacción	double
DEAL_COMMISSION	Comisión de transacción	double
DEAL_SWAP	Swap acumulado con el cierre	double

DEAL_PROFIT	Beneficio de transacción	double

Para la función HistoryDealGetString()

ENUM_DEAL_PROPERTY_STRING

Identificador	Descripción	Tipo
DEAL_SYMBOL	Símbolo de transacción	string
DEAL_COMMENT	Comentarios de transacción	string

Cada transacción se caracteriza por el tipo; los posibles valores se encuentran en la enumeración ENUM_DEAL_TYPE. Para obtener la información sobre el tipo de la transacción, utilice la función HistoryDealGetInteger() con el modificador DEAL_TYPE.

ENUM_DEAL_TYPE

Identificador	Descripción
DEAL_TYPE_BUY	Compra
DEAL_TYPE_SELL	Venta
DEAL_TYPE_BALANCE	Balance
DEAL_TYPE_CREDIT	Crédito
DEAL_TYPE_CHARGE	Cargas adicionales
DEAL_TYPE_CORRECTION	Corrección
DEAL_TYPE_BONUS	Bonos
DEAL_TYPE_COMMISSION	Comisiones adicionales
DEAL_TYPE_COMMISSION_DAILY	Comisión calculada al final de la jornada de trading
DEAL_TYPE_COMMISSION_MONTHLY	Comisión calculada al final del mes
DEAL_TYPE_COMMISSION_AGENT_DAILY	Comisión de agente calculada al final de la jornada de trading
DEAL_TYPE_COMMISSION_AGENT_MONTHLY	Comisión de agente calculada al final del mes
DEAL_TYPE_INTEREST	Calculo de intereses sobre fondos libres
DEAL_TYPE_BUY_CANCELED	Transacción de compra cancelada. Puede surgir la situación cuando una transacción de compra realizada anteriormente se cancele. En este caso el tipo de la operación realizada (DEAL_TYPE_BUY) se cambia a DEAL_TYPE_BUY_CANCELED, y su beneficio/pérdida se anula. El beneficio/pérdida producido/a anteriormente se carga/se quita de la cuenta por medio de una operación

	contable separada
DEAL_TYPE_SELL_CANCELED	Transacción de venta cancelada. Puede surgir la situación cuando una transacción de venta realizada anteriormente se cancele. En este caso el tipo de la operación realizada (DEAL_TYPE_SELL) se cambia a DEAL_TYPE_SELL_CANCELED, y su beneficio/pérdida se anula. El beneficio/pérdida producido/a anteriormente se carga/se quita de la cuenta por medio de una operación contable separada

Las transacciones se diferencian no sólo por sus tipos que se establecen de la enumeración ENUM_DEAL_TYPE, sino por el modo de cambiar la posición. Esto puede ser una simple apertura de posición, o acumulación de una posición abierta anteriormente (entrada en el mercado), cierre de posición con una transacción de dirección opuesta de un volumen correspondiente (salida del mercado), o bien, la vuelta de posición en caso cuando el volumen de transacción en dirección opuesta supera el volumen de la posición abierta anteriormente.

Todas estas situaciones se describen por los valores de la enumeración ENUM_DEAL_ENTRY. para obtener esta información, utilice la función HistoryDealGetInteger() con el modificador DEAL_ENTRY.

ENUM_DEAL_ENTRY

Identificador	Descripción
DEAL_ENTRY_IN	Entrada en el mercado
DEAL_ENTRY_OUT	Salida del mercado
DEAL_ENTRY_INOUT	Vuelta
DEAL_ENTRY_STATE	Rasgo de grabación de estatus

Tipos de transacciones comerciales

La actividad comercial se realiza mediante el envío de las ordenaciones de apertura de posiciones usando la función OrderSend(), también mediante las ordenaciones de colocación, modificación y eliminación de órdenes pendientes. Cada orden comercial contiene la especificación del tipo de la operación comercial solicitada. Las operaciones comerciales se describen en la enumeración ENUM_TRADE_REQUEST_ACTIONS.

ENUM_TRADE_REQUEST_ACTIONS

Identificador	Descripción
TRADE_ACTION_DEAL	Colocar una orden comercial de conclusión inmediata de un transacción según los parámetros especificados (colocar una orden de mercado)
TRADE_ACTION_PENDING	Colocar una orden comercial para la conclusión de una transacción bajo unas condiciones especificadas (orden pendiente)
TRADE_ACTION_SLTP	Modificar los valores Stop Loss y Take Profit de una posición abierta
TRADE_ACTION_MODIFY	Modificar los parámetros de una orden comercial colocada anteriormente
TRADE_ACTION_REMOVE	Eliminar una orden pendiente colocada anteriormente

Tipos de transacciones (transactions) comerciales

Como resultado de ejecución de ciertas acciones con la cuenta de trading su estado se cambia. A estas acciones les pertenecen:

- El envío de una solicitud comercial por parte de cualquier aplicación MQL5 en el terminal de cliente utilizando la función OrderSend y OrderSendAsync, con su posterior ejecución;
- El envío de una solicitud comercial a través de la interfaz gráfica del terminal y su posterior ejecución;
- El accionamiento de órdenes pendientes y órdenes Stop en el servidor;
- La ejecución de operaciones en el servidor de trading.

Como resultado de estas acciones, para la cuenta se ejecutan las transacciones comerciales:

- tramitación de la solicitud comercial;
- cambio de órdenes abiertas;
- cambio del historial de órdenes;
- cambio del historial de operaciones;
- cambio de posiciones.

Por ejemplo, al enviar una orden de compra, ésta se tramita, para la cuenta se crea una orden de compra correspondiente, se realiza la ejecución de la orden, su eliminación de la lista de las abiertas, se agrega al historial de órdenes, luego la operación correspondiente se agrega al historial, y se crea una posición nueva. Pues todas estas acciones son transacciones comerciales.

Para que el programador pueda monitorear las acciones realizadas respecto a la cuenta de trading, está prevista la función OnTradeTransaction. Este manejador permite recibir en la aplicación MQL5 las transacciones comerciales que han sido aplicadas a la cuenta. La descripción de una transacción comercial se envía en el primer parámetro de OnTradeTransaction a través de la estructura MqlTradeTransaction.

El tipo de la transacción comercial se envía en el parámetro type de la estructura MqlTradeTransaction. Los posibles tipos de transacciones comerciales se describen en la enumeración:

ENUM_TRADE_TRANSACTION_TYPE

Identificador	Descripción
TRADE_TRANSACTION_ORDER_ADD	Agregación de una nueva orden abierta.
TRADE_TRANSACTION_ORDER_UPDATE	Modificación de orden abierta. A estas modificaciones les corresponden no sólo los cambios explícitos por parte del terminal de cliente o servidor de trading, sino también alteraciones de su estado durante la colocación (por ejemplo, paso del estado ORDER_STATE_STARTED a ORDER_STATE_PLACED o de ORDER_STATE_PLACED a ORDER_STATE_PARTIAL etc.).
TRADE_TRANSACTION_ORDER_DELETE	Eliminación de la orden de la lista de órdenes

	abiertas. Una orden puede ser eliminada de las abiertas como resultado de colocación de la solicitud correspondiente, o bien una vez ejecutada (llena) y pasada al historial.
TRADE_TRANSACTION_DEAL_ADD	Agregación de una transacción (deal) al historial. Se hace como resultado de ejecución de la orden o realización de la operación con el balance de la cuenta.
TRADE_TRANSACTION_DEAL_UPDATE	Modificación de una transacción (deal) en el historial. Puede pasar que una transacción (deal) ejecutada antes se cambie en el servidor. Por ejemplo, la transacción (deal) fue modificada en el sistema externo de trading (bolsa) a donde había sido pasada por el broker.
TRADE_TRANSACTION_DEAL_DELETE	Eliminación de una transacción (deal) del historial. Puede pasar que una transacción (deal) ejecutada antes se elimine en el servidor. Por ejemplo, la transacción (deal) fue eliminada en el sistema externo de trading (bolsa) a donde había sido pasada por el broker.
TRADE_TRANSACTION_HISTORY_ADD	Agregación de una orden al historial como resultado de su ejecución o cancelación.
TRADE_TRANSACTION_HISTORY_UPDATE	Modificación de una orden que se encuentra en el historial de órdenes. Este tipo está previsto para ampliar la funcionalidad en la parte del servidor.
TRADE_TRANSACTION_HISTORY_DELETE	Eliminación de una orden del historial de órdenes. Este tipo está previsto para ampliar la funcionalidad en la parte del servidor.
TRADE_TRANSACTION_POSITION	Modificación de la posición que no está relacionada con la ejecución de la transacción (deal). Este tipo de transacción (transaction) quiere decir que la posición ha sido modificada en la parte del servidor de trading. La posición puede sufrir el cambio del volumen, precio de apertura, así como de los niveles Stop Loss y Take Profit. La información sobre los cambios se envía en la estructura MqlTradeTransaction usando el manejador OnTradeTransaction. La modificación de una posición (agregación, cambio o eliminación) como resultado de ejecución de la transacción (deal) no supone la aparición tras sí la transacción (transaction) TRADE_TRANSACTION_POSITION.

TRADE_TRANSACTION_REQUEST	Aviso de que la solicitud comercial ha sido procesada por el servidor, y el resultado de su procesamiento ha sido recibido. Para las transacciones de este tipo en la estructura MqlTradeTransaction hay que analizar sólo un campo - type (tipo de transacción). Para obtener la información adicional hay que analizar el segundo y el tercer parámetro de la función OnTradeTransaction (request y result).

En función del tipo de la transacción comercial, en la estructura MqlTradeTransaction que la describe, se rellenan diferentes parámetros. La descripción detallada de los datos traspasados se puede encontrar en el apartado "Estructura de transacción comercial".

Véase también

Estructura de transacción comercial, OnTradeTransaction

Tipos de órdenes en la profundidad de mercado

Para los instrumentos bursátiles está disponible la ventana "Profundidad de Mercado", donde se puede ver las actuales órdenes de compra y venta. Para cada orden se especifica la dirección de operación comercial deseada, volumen requerido y el precio solicitado.

Para recibir la información sobre el estado actual de la profundidad de mercado utilizando los medios del lenguaje MQL5, tenemos la función MarketBookGet(). Esta función coloca "screenshot de la profundidad de mercado" en la matriz de estructuras MqlBookInfo. Cada elemento de esta matriz contiene información en el campo *type* sobre la dirección de la orden, es el valor de la enumeración ENUM_BOOK_TYPE.

ENUM_BOOK_TYPE

Identificador	Descripción
BOOK_TYPE_SELL	Orden de venta
BOOK_TYPE_BUY	Orden de compra
BOOK_TYPE_SELL_MARKET	Solicitud de venta por el precio de mercado
BOOK_TYPE_BUY_MARKET	Solicitud de compra por el precio de mercado

Véase también

Estructuras y clases, Estructura de profundidad de mercado, Tipos de operaciones comerciales, Obtención de información de mercado

Constantes nombradas

Todas las constantes utilizadas en el lenguaje MQL5 pueden ser divididas en siguientes grupos:

- Macro substituciones predefinidas – los valores se substituyen durante la compilación;
- Constantes matemáticas – los valores de algunas expresiones matemáticas;
- Constantes de tipos numéricos – restricciones aplicadas a algunos tipos simples;
- Razones de deinicialización – descripción de las razones de deinicialización;
- Verificación del puntero a objeto – enumeración de los tipos de punteros devueltos por la función CheckPointer() ;
- Otras constantes – todas las demás constantes.

Macro substituciones predefinidas

Para facilitar la depuración y obtener la información sobre el funcionamiento de un programa mql5 están previstas las constantes-macros especiales predefinidas, cuyos valores se establecen en el momento de compilación. La manera más fácil de usarlas consiste en la devolución de sus valores mediante la función Print(), como se muestra en el ejemplo de abajo.

Constante	Descripción
__DATE__	Fecha de compilación del archivo sin hora (horas, minutos y segundos son iguales a 0)
__DATETIME__	Fecha y hora de compilación del archivo
__LINE__	Número de cadena en el código fuente, en la que se ubica esta macro
__FILE__	Nombre del archivo corriente compilado
__PATH__	Ruta absoluta hacia el archivo actual a compilar
__FUNCTION__	Nombre de la función, en cuyo cuerpo está ubicado la macro
__FUNCSIG__	Signatura de la función en cuyo cuerpo se encuentra la macro. El mostrar en el log la descripción completa de la función con tipos de parámetros puede ser útil durante la identificación de las funciones sobrecargadas
__MQ5BUILD__	Número build del compilador

Ejemplo:

```
#property copyright "Copyright © 2009, MetaQuotes Software Corp."
#property link      "http://www.metaquotes.net"
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- ejemplo de output de información durante la inicialización del Asesor Experto
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
//--- definición de intervalos entre los eventos de temporizador
   EventSetTimer(5);
//---
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- ejemplo de output de información durante la deinicialización del Asesor Experto
```

```
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
//---
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {

//--- output de información durante la llegada de del tick
   Print(" __MQ5BUILD__ = ",__MQ5BUILD__,"  __FILE__ = ",__FILE__);
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
   test1(__FUNCTION__);
   test2();
//---
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void test1(string par)
  {
//--- output de información dentro de la función
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__," par = ",par);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void test2()
  {
//--- output de información dentro de la función
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
void OnTimer()
  {
//---
   Print(" __FUNCTION__ = ",__FUNCTION__,"  __LINE__ = ",__LINE__);
   test1(__FUNCTION__);
  }
//+------------------------------------------------------------------+
```

Constantes matemáticas

Las constantes especiales que contienen valores están reservadas para algunas expresiones matemáticas. Se puede usar estas constantes en cualquier parte del programa mql5 en vez de calcular sus valores a través de las funciones matemáticas.

Constante	Descripción	Valor
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)	2.3025850929940456840 2
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)	0.70710678118654752440 1

Ejemplo:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- devolvemos los valores de las constantes
   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));
```

```
}
```

Constantes de tipos numéricos de datos

Cada tipo numérico simple sirve para un cierto grupo de tareas y permite optimizar el funcionamiento de un programa mql5, siempre y cuando se use correctamente. Para la mejor legibilidad del código y procesamiento correcto de los resultados de calculación, existen unas constantes que permiten recibir la información sobre las restricciones establecidas para uno u otro tipo de datos simples.

Constante	Descripción	Valor
CHAR_MIN	Valor mínimo que puede ser representado por el tipo char	-128
CHAR_MAX	Valor máximo que puede ser representado por el tipo char	127
UCHAR_MAX	Valor máximo que puede ser representado por el tipo uchar	255
SHORT_MIN	Valor mínimo que puede ser representado por el tipo short	-32768
SHORT_MAX	Valor máximo que puede ser representado por el tipo short	32767
USHORT_MAX	Valor máximo que puede ser representado por el tipo ushort	65535
INT_MIN	Valor mínimo que puede ser representado por el tipo int	-2147483648
INT_MAX	Valor máximo que puede ser representado por el tipo int	2147483647
UINT_MAX	Valor máximo que puede ser representado por el tipo uint	4294967295
LONG_MIN	Valor mínimo que puede ser representado por el tipo long	-9223372036854775808
LONG_MAX	Valor máximo que puede ser representado por el tipo long	9223372036854775807
ULONG_MAX	Valor máximo que puede ser representado por el tipo ulong	18446744073709551615
DBL_MIN	Valor mínimo positivo que puede ser representado por el tipo double	2.2250738585072014e-308
DBL_MAX	Valor máximo que puede ser representado por el tipo double	1.7976931348623158e+308
DBL_EPSILON	Valor mínimo que satisface la condición: 1.0+DBL_EPSILON != 1.0	2.2204460492503131e-016

DBL_DIG	Número de dígitos decimales significativos	15
DBL_MANT_DIG	Cantidad de bits en la mantisa	53
DBL_MAX_10_EXP	Valor decimal máximo del grado de exponente	308
DBL_MAX_EXP	Valor binario máximo del grado de exponente	1024
DBL_MIN_10_EXP	Valor decimal máximo del grado de exponente	(-307)
DBL_MIN_EXP	Valor binario mínimo del grado de exponente	(-1021)
FLT_MIN	Valor mínimo positivo que puede ser representado por el tipo float	1.175494351e-38
FLT_MAX	Valor máximo que puede ser representado por el tipo float	3.402823466e+38
FLT_EPSILON	Valor mínimo que satisface la condición: 1.0+FLT_EPSILON != 1.0	1.192092896e-07
FLT_DIG	Número de dígitos decimales significativos	6
FLT_MANT_DIG	Cantidad de bits en la mantisa	24
FLT_MAX_10_EXP	Valor decimal máximo del grado de exponente	38
FLT_MAX_EXP	Valor binario máximo del grado de exponente	128
FLT_MIN_10_EXP	Valor decimal mínimo del grado de exponente	-37
FLT_MIN_EXP	Valor binario mínimo del grado de exponente	(-125)

Ejemplo:

```
void OnStart()
  {
//--- devolvemos los valores de las constantes
   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);
  }
```

Razones de deinicialización

Los códigos de las razones de deinicialización del Asesor Experto devueltos por la función UninitializeReason(). Pueden tener cualquier de los siguientes valores:

Constante	Valor	Descripción
REASON_PROGRAM	0	Asesor Experto finalizó su trabajo llamando a la función ExpertRemove()
REASON_REMOVE	1	Programa ha sido eliminado del gráfico
REASON_RECOMPILE	2	Programa ha sido recompilado
REASON_CHARTCHANGE	3	Símbolo o período del gráfico ha sido modificado
REASON_CHARTCLOSE	4	Gráfico ha sido cerrado
REASON_PARAMETERS	5	Parámetros de entrada han sido cambiados por el usuario
REASON_ACCOUNT	6	Активирован другой счет либо произошло переподключение к торговому серверу вследствие изменения настроек счета
REASON_TEMPLATE	7	Una nueva plantilla del gráfico ha sido aplicada
REASON_INITFAILED	8	Este valor significa que el manejador OnInit() ha devuelto un valor no nulo
REASON_CLOSE	9	El terminal ha sido cerrado

El código de la razón de deinicialización se pasa también como un parámetro de la función predefinida OnDeinit(const int reason).

Por ahora los indicadores sólo admiten el código 1(REASON_REMOVE) y 2(REASON_RECOMPILE).

Ejemplo:

```
//+------------------------------------------------------------------+
//| get text description                                             |
//+------------------------------------------------------------------+
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)
  {
//--- El primer modo de obtener el código de la razón de deinicialización
   Print(__FUNCTION__,"_Código de la razón de deinicialización = ",reason);
//--- El segundo modo de obtener el código de la razón de deinicialización
   Print(__FUNCTION__,"_UninitReason = ",getUninitReasonText(_UninitReason));
  }
```

Verificación del puntero a objeto

La función CheckPointer() sirve para comprobar el tipo del puntero a objeto. Esta función devuelve el valor de la enumeración ENUM_POINTER_TYPE. En caso de usar un puntero incorrecto, la ejecución del programa se detendrá inmediatamente.

Los objetos creados por el operador new son del tipo POINTER_DYNAMIC. Sólo para estos punteros se puede y se debe usar el operador de eliminación delete().

Todos los demás punteros tienen el tipo POINTER_AUTOMATIC, lo que significa que este objeto ha sido creado automáticamente por el entorno del programa mql5. Estos objetos se eliminan después de ser usados también de una manera automática.

ENUM_POINTER_TYPE

Constante	Descripción
POINTER_INVALID	Puntero incorrecto
POINTER_DYNAMIC	Puntero a objeto que ha sido creado por el operador new
POINTER_AUTOMATIC	Puntero a cualquier objeto que ha sido creado automáticamente (sin usar new())

Véase también

Errores de ejecución, Operador de eliminación de objeto delete, CheckPointer()

Otras constantes

La constante CLR_NONE sirve para indicar la falta del color, es decir, el objeto gráfico o serie gráfica de un indicador no serán mostrados. Esta constante no ha entrado en la lista de constantes de colores Web pero se puede usarla en cualquier parte donde se requiere indicar un color.

La constante INVALID_HANDLE puede ser usada durante la depuración de los manejadores de archivos (véase FileOpen() y FileFindFirst()).

Constante	Descripción	Valor
CHARTS_MAX	La cantidad máxima posible de los gráficos abiertos al mismo tiempo en el terminal	100
clrNONE	Ausencia de color	-1
EMPTY_VALUE	Valor vacío en el búfer de indicadores	DBL_MAX
INVALID_HANDLE	Manejador incorrecto	-1
IS_DEBUG_MODE	Indica que un programa mql5 se encuentra en el modo de depuración	true en el modo de depuración, de lo contrario false
IS_PROFILE_MODE	Indica que un programa mql5 se encuentra en el modo de perfilación	en modo de perfilación no es igual a cero, de lo contrario 0
NULL	Cero de cualquier tipo	0
WHOLE_ARRAY	Significa el número de elementos que se quedan hasta el final del array, es decir, el array entero será procesado	-1
WRONG_VALUE	La constante puede convertirse implícitamente al tipo de cualquier enumeración.	-1

La constante EMPTY_VALUE suele corresponder a los valores de los indicadores que no se muestran en el gráfico. Por ejemplo, para el indicador built-in Standard Deviation con el período 20, la línea para las primeras 19 barras en el historial no se muestra en el gráfico. Si creamos el manejador de este indicador usando la función iStdDev() y copiamos en el array los valores del indicador para estas baras a través de CopyBuffer(), entonces precisamente estos valores serán iguales a EMPTY_VALUE.

Nosotros mismos podemos especificar nuestro propio valor vacío del indicador en el indicador personalizado, en este caso el indicador no debería mostrarse en el gráfico. Con este fin se usa la función PlotIndexSetDouble() con el modificador PLOT_EMPTY_VALUE.

La constante NULL puede ser asignada a una variable de cualquier tipo simple o a un puntero a objeto de estructura o clase. La asignación de NULL a una variable de cadena significa la deinicialización

completa de esta variable.

La constante WRONG_VALUE sirve para los casos cuando hace falta devolver el valor de una enumeración, y éste tiene que ser un valor erróneo. Por ejemplo, cuando tenemos que informar que un valor devuelto es un valor de esta enumeración. Como ilustración vamos a considerar una función CheckLineStyle() que devuelve el estilo de la línea para un objeto especificado por su nombre. Si el resultado de comprobación del estilo por la función ObjectGetInteger() es true, entonces será devuelto el valor de la enumeración ENUM_LINE_STYLE, de lo contrario se devuelve WRONG_VALUE.

```
void OnStart()
  {
   if(CheckLineStyle("MyChartObject")==WRONG_VALUE)
      printf("Error line style getting.");
  }
//+------------------------------------------------------------------+
//| devuelve el estilo de la línea de un objeto especificado por su nombre          |
//+------------------------------------------------------------------+
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);
  }
```

La constante WHOLE_ARRAY está destinada para las funciones que requieren la especificación de la cantidad de elementos en los arrays procesados:

- ArrayCopy();
- ArrayMinimum();
- ArrayMaximum();
- FileReadArray();
- FileWriteArray().

Si hace falta especificar que es necesario procesar todos los valores del array desde la posición especificada hasta el final, será suficiente indicar el valor WHOLE_ARRAY.

La constante IS_PROFILE_MODE permite cambiar el trabajo del programa para la correcta recopilación de la información en el modo de perfilación. La perfilación permite medir el tiempo de ejecución de ciertos fragmentos del programa (normalmente son las funciones), así como contar el número de estas llamadas. Para una correcta obtención de información sobre el tiempo de ejecución en el modo de perfilación se puede desactivar las llamadas a la función Sleep() como se muestra en el ejemplo:

```
//--- Sleep puede influir (desfigurar) considerablemente en el resultado de la perfil
if(!IS_PROFILE_MODE) Sleep(100); // prohibimos la llamada a Sleep() en el modo de per
```

El valor de la constante IS_PROFILE_MODE se establece por el compilador en el momento de compilación, mientras que en el modo convencional se pone igual a cero. Cuando el programa se inicia en el modo de perfilación, se lleva a cabo una compilación especial, y en este caso IS_PROFILE_MODE se sustituye con un valor distinto a cero.

La constante IS_DEBUG_MODE será útil cuando es necesario cambiar un poco el trabajo de un

programa mql5 en el modo de depuración. Por ejemplo, durante la depuración surge la necesidad de mostrar la información adicional de depuración en el registro (log) del terminal o crear los objetos gráficos auxiliares en un gráfico.

El ejemplo de abajo crea un objeto Label y define su descripción y color dependiendo del régimen en el que se ejecute el script. Para iniciar un script en el modo de depuración en MetaEditor, presione el botón **F5**. Si iniciamos el script desde la ventana del navegador en el terminal, el color y el texto del objeto Label serán diferentes.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                              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());
      //--- creamos el objeto Label
      ObjectCreate(0,label_name,OBJ_LABEL,0,0,0);
      //--- establecemos la coordinada X
      ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,200);
      //--- establecemos la coordinada Y
      ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,300);
      ResetLastError();
      if(IS_DEBUG_MODE) // modo de depuración
        {
         //--- mostramos el mensaje sobre el modo de ejecución del script
         ObjectSetString(0,label_name,OBJPROP_TEXT,"DEBUG MODE");
         //--- establecemos el color del texto como rojo
         if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,clrRed))
            Print("Fallo al determinar el color. Error ",GetLastError());
        }
      else              // modo operacional
        {
         ObjectSetString(0,label_name,OBJPROP_TEXT,"RELEASE MODE");
         //--- fijamos el color invisible del texto
         if(!ObjectSetInteger(0,label_name,OBJPROP_COLOR,CLR_NONE))
            Print("Fallo al determinar el color. Error ",GetLastError());
        }
      ChartRedraw();
      DebugBreak();     // si nos encontramos en el modo de depuración, aquí sucederá
     }
  }
```

Véase también

DebugBreak, Información sobre el programa MQL5 en ejecución

Estructuras de datos

En MQL5 hay 8 estructuras predefinidas que sirven para el almacenamiento y traspaso de información auxiliar:

- MqlDateTime sirve para representar fecha y hora;
- MqlParam permite pasar los parámetros de entrada durante la creación de un manejador (handle) de indicador utilizando la función IndicatorCreate();
- MqlRates se usa para facilitar la información sobre los datos históricos que contienen el precio, volumen y spread;
- MqlBookInfo se usa para obtener información reflejada en la profundidad de mercado (ventana de cotizaciones);
- MqlTradeRequest se usa para crear una orden comercial durante las operaciones comerciales;
- MqlTradeResult contiene la respuesta del servidor comercial a una orden comercial mandada por la función OrderSend();
- MqlTradeTransaction contiene la descripción de transacción comercial;
- MqlTick sirve para la obtención rápida de la información más requerida sobre los precios actuales.

MqlDateTime

La estructura de la fecha contiene ocho campos del tipo int.

```
struct MqlDateTime
  {
   int year;            // año
   int mon;             // mes
   int day;             // día
   int hour;            // hora
   int min;             // minutos
   int sec;             // segundos
   int day_of_week;     // día de la semana (0-domingo, 1-lunes, ... ,6-sábado)
   int day_of_year;     // número del día del año (el primero de enero tiene el número 0)
  };
```

Nota

Número del día del año day_of_year del año bisiesto, empezando desde Marzo, será diferente del número del día del año en el año no bisiesto.

Ejemplo:

```
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);
  }
/*  Resultado
   01.03.2008, day of year = 60
   01.03.2009, day of year = 59
*/
```

Véase también

TimeToStruct, Estructuras y clases

Estructura de parámetros de entrada de indicador (MqlParam)

La estructura MqlParam ha sido diseñada especialmente para traspasar los parámetros de entrada cuando se crea el manejador del indicador técnico usando la función IndicatorCreate().

```
struct MqlParam
  {
   ENUM_DATATYPE     type;                    // tipo del parámetro de entrada, valor de la enumerac
   long              integer_value;           // campo para almacenar valores de números enteros
   double            double_value;            // campo para almacenar valores double o float
   string            string_value;            // campo para almacenar valores del tipo string
  };
```

Todos los parámetros de entrada se transmiten en forma de una matriz del tipo MqlParam, el campo *type* de cada uno de los elementos de esta matriz especifica el tipo de datos que transmite dicho elemento. Previamente hay que colocar los valores de parámetros del indicador en los campos correspondientes para cada elemento (en *integer_value*, en *double_value* o en *string_value*), dependiendo de qué valor de la enumeración ENUM_DATATYPE figura en el campo *type*.

Si el valor IND_CUSTOM en el tercer parámetro se pasa a la función IndicatorCreate() como el tipo de indicador, entonces el primer elemento de la matriz de los parámetros de entrada debe tener el campo *type* con el valor TYPE_STRING de la enumeración ENUM_DATATYPE, y el campo *string_value* tiene que contener el nombre del indicador personalizado.

MqlRates

Esta estructura sirve para almacenar la información sobre los precios, volúmenes y spread.

```
struct MqlRates
  {
   datetime time;         // hora del inicio del período
   double   open;         // precio de apertura
   double   high;         // precio máximo durante el período
   double   low;          // precio mínimo durante el período
   double   close;        // precio de cierre
   long     tick_volume;  // volumen de tick
   int      spread;       // spread
   long     real_volume;  // volumen de stock
  };
```

Ejemplo:

```
void OnStart()
  {
   MqlRates rates[];
   int copied=CopyRates(NULL,0,0,100,rates);
   if(copied<=0)
      Print("Fallo al copiar los datos de precios ",GetLastError());
   else Print("Se ha copiado ",ArraySize(rates)," barras");
  }
```

Véase también

CopyRates, Acceso a series temporales

MqlBookInfo

Esta estructura proporciona la información sobre la profundidad de mercado.

```
struct MqlBookInfo
  {
   ENUM_BOOK_TYPE    type;       // tipo de orden desde la enumeración ENUM_BOOK_TYPE
   double            price;      // precio
   long              volume;     // volumen
  };
```

Nota

La estructura MqlBookInfo es predefinida, por eso no hace falta declarar y describirla. Para utilizar la estructura, será suficiente declarar la variable de este tipo.

La profundidad de mercado está disponible sólo para algunos instrumentos financieros.

Ejemplo:

```
   MqlBookInfo priceArray[];
   bool getBook=MarketBookGet(NULL,priceArray);
   if(getBook)
     {
      int size=ArraySize(priceArray);
      Print("MarketBookInfo para ",Symbol());
     }
   else
     {
      Print("Fallo al recibir el contenido de la profundidad de mercado para el símbo
     }
```

Véase también

MarketBookAdd, MarketBookRelease, MarketBookGet, Tipos de órdenes en profundidad de mercado, Tipos de datos

Estructura de solicitud comercial (MqlTradeRequest)

La interacción entre el terminal de cliente y el servidor comercial con el fin de ejecutar las operaciones de colocación de las órdenes se realiza a través de las solicitudes comerciales. La solicitud comercial está representada por la estructura especial predefinida MqlTradeRequest que contiene todos los campos necesarios para celebrar las transacciones comerciales. El resultado de procesamiento de una solicitud está representado por la estructura MqlTradeResult.

```
struct MqlTradeRequest
  {
   ENUM_TRADE_REQUEST_ACTIONS    action;           // Tipo de acción que se ejecuta
   ulong                         magic;            // ID del Asesor Experto (identificador magic num
   ulong                         order;            // Ticket de la orden
   string                        symbol;           // Nombre del instrumento comercial
   double                        volume;           // Volumen solicitado de la transacción en lotes
   double                        price;            // Precio
   double                        stoplimit;        // Nivel StopLimit de la orden
   double                        sl;               // Nivel Stop Loss de la orden
   double                        tp;               // Nivel Take Profit de la orden
   ulong                         deviation;        // Desviación máxima aceptable del precio solicit
   ENUM_ORDER_TYPE               type;             // Tipo de orden
   ENUM_ORDER_TYPE_FILLING       type_filling;     // Tipo de ejecución de orden
   ENUM_ORDER_TYPE_TIME          type_time;        // Tipo de orden por su plazo de ejecución
   datetime                      expiration;       // Plazo de expiración de orden (para las órdenes
   string                        comment;          // Comentarios sobre la orden
  };
```

Descripción de campos

Campo	Descripción
action	Tipo de operación comercial. El valor puede ser uno de los valores de la enumeración ENUM_TRADE_REQUEST_ACTIONS
magic	Identificador del Asesor Experto. Permite organizar el procesamiento analítico de las órdenes comerciales. Cada Asesor Experto puede establecer su propio identificador personal único a la hora de mandar una solicitud comercial.
order	Ticket de la orden. Se necesita para modificar las órdenes pendientes.
symbol	Nombre del instrumento financiero (símbolo) para el que se coloca una orden. No se necesita para modificar las órdenes y cerrar las posiciones.
volume	Volumen solicitado de la transacción en lotes. El valor real del volumen dependerá del tipo de ejecución de la orden.
price	Precio. Al alcanzarlo, la orden tiene que ejecutarse. Las órdenes del mercado de

	símbolos, cuyo tipo de ejecución es "Market Execution" (SYMBOL_TRADE_EXECUTION_MARKET), del tipo TRADE_ACTION_DEAL, no requieren la especificación del precio.
stoplimit	Precio según el cual será colocado la orden pendiente Limit, cuando el precio alcance el valor price (esta condición es obligatoria). Hasta entonces la orden pendiente no se introduce en el sistema.
sl	Precio que activará la orden Stop Loss, si el precio se mueve en la dirección desfavorable
tp	Precio que activará la orden Take Profit. si el precio se mueve en la dirección favorable
deviation	Desviación máxima aceptable del precio solicitado, se especifica en puntos
type	Tipo de la orden. Su valor puede ser uno de los valores de la enumeración ENUM_ORDER_TYPE
type_filling	Tipo de ejecución de la orden. Su valor puede ser uno de los valores de ENUM_ORDER_TYPE_FILLING
type_time	Tipo de orden por su plazo de ejecución. Su valor puede ser uno de los valores de ENUM_ORDER_TYPE_TIME
expiration	Plazo de expiración de la orden (para las órdenes del tipo ORDER_TIME_SPECIFIED)
comment	Comentarios sobre la orden

Para dar las órdenes de ejecución de las operaciones comerciales es necesario usar la función OrderSend(). Para cada operación comercial hay que indicar los campos obligatorios, también se puede rellenar campos opcionales. En total hay siete formas de enviar una solicitud comercial:

Request Execution

Es una orden comercial para abrir una posición en el régimen Request Execution (régimen de actividad comercial sobre solicitud de precios actuales). Se requiere especificar 9 campos:

- action
- symbol
- volume
- price
- sl
- tp
- deviation
- type
- type_filling

Además se puede definir los valores de los campos magic y comment.

Instant Execution

Es una orden comercial para abrir una posición en el régimen Instant Execution (régimen de actividad comercial a base de los precios corrientes). Se requiere especificar 9 campos:

- action
- symbol
- volume
- price
- sl
- tp
- deviation
- type
- type_filling

Además se puede definir los valores de los campos magic comment.

Market Execution

Es una orden comercial para abrir una posición en el régimen Market Execution (régimen de ejecución de las órdenes comerciales en el mercado). Se requiere especificar 5 campos:

- action
- symbol
- volume
- type
- type_filling

Además se puede definir los valores de los campos magic y comment.

Exchange Execution

Es una orden comercial para abrir una posición en el régimen Exchange Execution (modo de ejecución de órdenes comerciales por bolsa). Se requiere especificar 5 campos:

- action
- symbol
- volume
- type
- type_filling

Además se puede definir los valores de los campos magic y comment.

SL & TP Modification

Es una orden comercial para modificar los niveles StopLoss y/o TakeProfit. Se requiere especificar 4 campos:

- action
- symbol
- sl
- tp

Pending Order

Es una orden comercial para colocar una orden pendiente. Se requiere especificar 11 campos:

- action
- symbol
- volume
- price
- stoplimit
- sl
- tp
- type
- type_filling
- type_time
- expiration

Además se puede definir los valores de los campos magic y comment.

Modify Pending Order

Es una orden comercial para modificar los precios de una orden pendiente. Se requiere especificar 7 campos:

- action
- order
- price
- sl
- tp
- type_time
- expiration

Delete Pending Order

Es una orden comercial para eliminar una orden pendiente. Se requiere especificar 2 campos:

- action
- order

Véase también

Estructuras y clases, Funciones comerciales, Propiedades de órdenes

Estructura de comprobación de solicitud comercial (MqlTradeCheckResult)

Antes de enviar una solicitud para una operación comercial al servidor comercial, se recomienda efectuar comprobaciones de su viabilidad. Dicha comprobación se realiza usando la función OrderCheck() a la que se pasa la solicitud a comprobar y también la variable del tipo de estructura MqlTradeCheckResult. Precisamente en esta variable será reflejado el resultado del chequeo realizado.

```
struct MqlTradeCheckResult
  {
   uint         retcode;             // Código de respuesta
   double       balance;             // Balance después de realizar la transacción
   double       equity;              // Capital privado después de realizar la transacción
   double       profit;              // Beneficio flotante
   double       margin;              // Requerimientos de margen
   double       margin_free;         // Margen libre
   double       margin_level;        // Nivel del margen
   string       comment;             // Comentarios sobre el código de respuesta (descripción del er
  };
```

Descripción de campos

Campo	Descripción
retcode	Código de retorno
balance	Balance que se queda tras la ejecución de la operación comercial
equity	Valor de fondos propios que se obtiene tras la ejecución de la operación comercial
profit	Valor del beneficio flotante que se obtiene tras la ejecución de la operación comercial
margin	Margen necesario para la operación comercial
margin_free	Fondos propios que se quedan disponibles después de realizar la operación comercial
margin_level	Nivel del margen que va a establecerse después de realizar la operación comercial
comment	Comentario sobre el código de respuesta, descripción del error en su caso

Véase también

Estructura de solicitud comercial, Estructura para obtención de precios actuales, OrderSend, OrderCheck

Estructura de resultado de solicitud comercial (MqlTradeResult)

Respondiendo a una solicitud comercial acerca de colocación de una orden en el sistema comercial, el servidor comercial devuelve los datos que contienen la información sobre el resultado de procesamiento de la solicitud comercial en forma de la estructura especial predefinida MqlTradeResult.

```
struct MqlTradeResult
  {
   uint     retcode;          // Código del resultado de operación
   ulong    deal;             // Ticket de transacción, si está concluida
   ulong    order;            // Ticket de la orden, si está colocada
   double   volume;           // Volumen de la transacción confirmado por el corredor
   double   price;            // Precio en la transacción confirmada por el corredor
   double   bid;              // Precio actual de la oferta en el mercado (precios recuota)
   double   ask;              // Precio actual de la demanda en el mercado (precios recuota)
   string   comment;          // Comentarios del corredor acerca de la operación (por defecto se rel
   uint     request_id;       // El terminal pone el identificador de la solicitud a la hora de envi
  };
```

Descripción de campos

Campo	Descripción
retcode	Código de retorno del servidor comercial
deal	Ticket de la transacción, si está concluida. Se comunica al ejecutar la operación comercial TRADE_ACTION_DEAL
order	Ticket de la orden, si está colocada. Se comunica al ejecutar la operación comercial TRADE_ACTION_PENDING
volume	Volumen de la transacción confirmado por el corredor. Depende del tipo de ejecución de la orden
price	Precio en la transacción confirmada por el corredor. Depende del campo *deviation* en la solicitud comercial y/o del tipo de la operación comercial
bid	Precio actual de la oferta en el mercado (precios recuota)
ask	Precio actual de la demanda en el mercado (precios recuota)
comment	Comentarios del corredor acerca de la operación (por defecto se rellena con la descripción de la operación)
request_id	Identificador de solicitud puesto por el terminal al inviarla al servidor de trading

El resultado de la operación comercial se devuelve en una variable del tipo MqlTradeResult la que se pasa como segundo parámetro a la función OrderSend() para realizar las operaciones comerciales.

El terminal registra el identificador de la solicitud en el campo request_id a la hora de enviarla al servidor de trading por medio de las funciones OrdersSend() y OrderSendAsync(). El terminal recibe los mensajes de parte del servidor de trading sobre las transacciones comerciales realizadas y las pasa para el procesamiento a la función OnTradeTransaction() que contiene como parámetro:

- descripción de la misma transacción comercial en la estructura MqlTradeTransaction;
- descripción de la solicitud comercial que ha sido enviada desde la función OrderSend() o OrdersSendAsync(). El terminal envía el identificador de la solicitud al servidor de trading, mientras que la misma solicitud y su request_id se guardan en la memoria del terminal;
- resultado de la ejecución de la solicitud comercial en forma de la estructura MqlTradeResult donde el campo request_id contiene el identificador de esta solicitud.

La función OnTradeTransaction() obtiene tres parámetros de entrada, pero los dos últimos parámetros tiene sentido analizarlos sólo para las transacciones comerciales que tienen el tipo TRADE_TRANSACTION_REQUEST. En todos los demás casos, los datos sobre la solicitud comercial y el resultado de su ejecución no se rellenan. El ejemplo del análisis de los parámetros se muestra en el apartado Estructura de transacción comercial.

La puesta del identificador request_id para la solicitud comercial por parte del terminal a la hora de enviarla al servidor está destinada en primer lugar para el trabajo con la función asincrónica OrderSendAsync(). Este identificador permite vincular la acción ejecutada (llamada a la función OrderSend o OrderSendAsync) con el resultado de esta acción que se traspasa en OnTradeTransaction().

Ejemplo:

```
//+------------------------------------------------------------------+
//|  Envío de una solicitud comercial con el procesamiento del resultado     |
//+------------------------------------------------------------------+
bool MyOrderSend(MqlTradeRequest request,MqlTradeResult result)
  {
//--- pongamos el código del último error a cero
   ResetLastError();
//--- enviamos la solicitud
   bool success=OrderSend(request,result);
//--- si ha fallado, vamos a intentar averiguar porqué
   if(!success)
     {
      int answer=result.retcode;
      Print("TradeLog:Trade request failed. Error = ",GetLastError());
      switch(answer)
        {
         //--- recuota
         case 10004:
           {
            Print("TRADE_RETCODE_REQUOTE");
            Print("request.price = ",request.price,"   result.ask = ",
                  result.ask," result.bid = ",result.bid);
            break;
           }
         //--- la orden no ha sido aceptada por el servidor
         case 10006:
           {
            Print("TRADE_RETCODE_REJECT");
            Print("request.price = ",request.price,"   result.ask = ",
                  result.ask," result.bid = ",result.bid);
            break;
           }
         //--- precio incorrecto
         case 10015:
           {
            Print("TRADE_RETCODE_INVALID_PRICE");
            Print("request.price = ",request.price,"   result.ask = ",
                  result.ask," result.bid = ",result.bid);
            break;
           }
         //---  SL y/o TP incorrecto(s)
         case 10016:
           {
            Print("TRADE_RETCODE_INVALID_STOPS");
            Print("request.sl = ",request.sl," request.tp = ",request.tp);
            Print("result.ask = ",result.ask," result.bid = ",result.bid);
            break;
           }
         //--- volumen incorrecto
         case 10014:
           {
            Print("TRADE_RETCODE_INVALID_VOLUME");
            Print("request.volume = ",request.volume,"   result.volume = ",
                  result.volume);
            break;
           }
         //--- falta dinero para esta operación comercial
         case 10019:
           {
            Print("TRADE_RETCODE_NO_MONEY");
            Print("request.volume = ",request.volume,"   result.volume = ",
```

```
                  result.volume,"   result.comment = ",result.comment);
            break;
           }
         //--- alguna otra razón, mostramos el código de respuesta del servidor
         default:
           {
            Print("Other answer = ",answer);
           }
        }
      //--- notificamos devolviendo false sobre el resultado fallido de la solicitud
      return(false);
     }
//--- OrderSend() ha devuelto true - repetimos la respuesta
   return(true);
  }
```

Estructura de transacción comercial (MqlTradeTransaction)

Como resultado de ejecución de ciertas acciones con la cuenta de trading su estado se cambia. A estas acciones les pertenecen:

- El envío de una solicitud comercial por parte de cualquier aplicación MQL5 en el terminal de cliente utilizando la función OrderSend y OrderSendAsync, con su posterior ejecución;
- El envío de una solicitud comercial a través de la interfaz gráfica del terminal y su posterior ejecución;
- El accionamiento de órdenes pendientes y órdenes Stop en el servidor;
- La ejecución de operaciones en el servidor de trading.

Como resultado de estas acciones, para la cuenta se ejecutan las transacciones comerciales:

- tramitación de la solicitud comercial;
- cambio de órdenes abiertas;
- cambio del historial de órdenes;
- cambio del historial de operaciones;
- cambio de posiciones.

Por ejemplo, al enviar una orden de compra, ésta se tramita, para la cuenta se crea una orden de compra correspondiente, se realiza la ejecución de la orden, su eliminación de la lista de las abiertas, se agrega al historial de órdenes, luego la operación correspondiente se agrega al historial, y se crea una posición nueva. Pues todas estas acciones son transacciones comerciales.

Para recibir las transacciones comerciales que se aplican a la cuenta, en MQL5 ha sido diseñado un manejador especial OnTradeTransaction(). En el primer parámetro de este manejador se traspasa la estructura MqlTradeTransaction que describe las transacciones comerciales.

```
struct MqlTradeTransaction
  {
   ulong                     deal;             // Ticket de la operación
   ulong                     order;            // Ticket de la orden
   string                    symbol;           // Nombre del instrumento financiero
   ENUM_TRADE_TRANSACTION_TYPE type;           // Tipo de transacción comercial
   ENUM_ORDER_TYPE           order_type;       // Tipo de la orden
   ENUM_ORDER_STATE          order_state;      // Estado de la orden
   ENUM_DEAL_TYPE            deal_type;        // Tipo de la operación
   ENUM_ORDER_TYPE_TIME      time_type;        // Tipo de la orden según el tiempo de ejecución
   datetime                  time_expiration;  // Plazo de vencimiento de la orden
   double                    price;            // Precio
   double                    price_trigger;    // Precio de activación de la orden stop limitada
   double                    price_sl;         // Nivel Stop Loss
   double                    price_tp;         // Nivel Take Profit
   double                    volume;           // Volumen en lotes
  };
```

Descripción de campos

Campo	Descripción
deal	Ticket de la operación.

order	Ticket de la orden.
symbol	Nombre del instrumento financiero para el que se realiza la transacción.
type	Tipo de transacción comercial. El valor puede ser uno de los valores de la enumeración ENUM_TRADE_TRANSACTION_TYPE.
order_type	Tipo de orden comercial. El valor puede ser uno de los valores de la enumeración ENUM_ORDER_TYPE.
order_state	Estado de orden comercial. El valor puede ser uno de los valores de la enumeración ENUM_ORDER_STATE.
deal_type	Tipo de operación. El valor puede ser uno de los valores de la enumeración ENUM_DEAL_TYPE.
type_time	Tipo de la orden según su expiración. El valor puede ser uno de los valores de la enumeración ENUM_ORDER_TYPE_TIME.
time_expiration	Plazo de expiración de la orden pendiente (para las órdenes del tipo ORDER_TIME_SPECIFIED y ORDER_TIME_SPECIFIED_DAY).
price	Precio. En función del tipo de la transacción comercial puede ser el precio de la orden, operación o posición.
price_trigger	Precio stop (precio de activación) de la orden stop limitada (ORDER_TYPE_BUY_STOP_LIMIT y ORDER_TYPE_SELL_STOP_LIMIT).
price_sl	Precio Stop Loss. En función del tipo de la transacción comercial puede referirse al precio de la orden, operación o posición.
price_tp	Precio Take Profit. En función del tipo de la transacción comercial puede referirse al precio de la orden, operación o posición.
volume	Volumen en lotes. En función del tipo de la transacción comercial puede referirse al volumen actual de la orden, volumen de la operación o volumen de la posición.

El parámetro determinante para el análisis de una transacción que llega es su tipo que figura en el campo **type**. Por ejemplo, si la transacción es del tipo TRADE_TRANSACTION_REQUEST (el resultado de procesamiento de la solicitud por parte del servidor ha sido recibido), entonces la estructura tiene sólo un campo rellenado **type**, los demás no hace falta analizar. En este caso se puede realizar el análisis de dos campos adicionales **request** y **result** que se pasan al manejador OnTradeTransaction(), tal como se muestra en el ejemplo de abajo.

Teniendo información sobre el tipo de la operación comercial, se puede tomar la decisión sobre el análisis del estado actual de la orden, posición y transacciones (deals) en la cuenta de trading. Hay que tener en cuenta que una solicitud comercial enviada del terminal al servidor puede provocar varias transacciones (transactions) comerciales, cuya orden de llegada al terminal no se garantiza.

La estructura MqlTradeTransaction se llena de una manera diferente en función del tipo de transacción comercial (ENUM_TRADE_TRANSACTION_TYPE):

TRADE_TRANSACTION_ORDER_* y TRADE_TRANSACTION_HISTORY_*

Para las transacciones comerciales que conciernen el procesamiento de las órdenes abiertas (TRADE_TRANSACTION_ORDER_ADD, TRADE_TRANSACTION_ORDER_UPDATE y TRADE_TRANSACTION_ORDER_DELETE) e historial de órdenes (TRADE_TRANSACTION_HISTORY_ADD, TRADE_TRANSACTION_HISTORY_UPDATE, TRADE_TRANSACTION_HISTORY_DELETE), en la estructura MqlTradeTransaction se llenan los siguientes campos:

- order - ticket de la orden;
- symbol - nombre del instrumento financiero en la orden;
- type - tipo de transacción (transaction) comercial;
- order_type - tipo de la orden;
- orders_state - estado actual de la orden;
- time_type - tipo de vencimiento de la orden;
- time_expiration - tiempo de expiración de la orden (para las órdenes con el tipo de vencimiento ORDER_TIME_SPECIFIED y ORDER_TIME_SPECIFIED_DAY);
- price - precio de la orden especificado por el cliente;
- price_trigger - precio stop de activación de la orden stop limitada (sólo para ORDER_TYPE_BUY_STOP_LIMIT y ORDER_TYPE_SELL_STOP_LIMIT);
- price_sl - precio Stop Loss de la orden (se rellena si está especificado en la orden);
- price_tp - precio Take Profit de la orden (se rellena si está especificado en la orden);
- volume - volumen actual de la orden (no ejecutado). El volumen inicial de la orden se puede conocer del historial de órdenes utilizando la función HistoryOrders*.

TRADE_TRANSACTION_DEAL_*

Para las transacciones (transactions) comerciales que conciernen el procesamiento de las operaciones (deals) (TRADE_TRANSACTION_DEAL_ADD, TRADE_TRANSACTION_DEAL_UPDATE y TRADE_TRANSACTION_DEAL_DELETE), en la estructura MqlTradeTransaction se llenan los siguientes campos:

- deal - ticket de la operación (deal);
- order - ticket de la orden a base de la cual ha sido realizada la operación (deal);
- symbol - nombre del instrumento financiero en la operación (deal);
- type - tipo de transacción (transaction) comercial;
- deal_type - tipo de la operación (deal);
- Precio — precio por el que ha sido realizada la operación (deal);
- price_sl - precio Stop Loss (se rellena si está especificado en la orden a base de la cual ha sido realizada la operación (deal));
- price_tp - precio Take Profit (se rellena si está especificado en la orden a base de la cual ha sido realizada la operación (deal));
- Volumen — volumen de la operación (deal) en lotes.

TRADE_TRANSACTION_POSITION

Para las transacciones (transactions) comerciales que conciernen las modificaciones de posiciones no relacionadas con la ejecución de las operaciones (deals) (TRADE_TRANSACTION_POSITION), en la estructura MqlTradeTransaction se llenan los siguientes campos:

- symbol - nombre del instrumento financiero de la posición;
- type - tipo de transacción (transaction) comercial;
- deal_type - tipo de posición (DEAL_TYPE_BUY o DEAL_TYPE_SELL);
- price - precio medio ponderado de la apertura de la posición;
- price_sl - precio Stop Loss;
- price_tp - precio Take Profit;
- Volumen — volumen de la posición en lotes si no ha sido modificado.

> La modificación de una posición (agregación, cambio o eliminación) como resultado de ejecución de la transacción (deal) no supone la aparición tras sí la transacción (transaction) TRADE_TRANSACTION_POSITION.

TRADE_TRANSACTION_REQUEST

Para las transacciones (transactions) comerciales que describen el hecho de que la solicitud comercial haya sido procesada por el servidor y que el resultado de su procesamiento haya sido recibido (TRADE_TRANSACTION_REQUEST), en la estructura MqlTradeTransaction se rellena sólo un campo:

- type - tipo de transacción (transaction) comercial;

> Para las transacciones de este tipo hay que analizar sólo un campo - type (tipo de transacción comercial). Para obtener la información adicional hay que analizar el segundo y el tercer parámetro de la función OnTradeTransaction (request y result).

Ejemplo:

```
input int MagicNumber=1234567;

//--- activamos la clase de trading CTrade y declaramos la variable de este tipo
#include <Trade\Trade.mqh>
CTrade trade;
//--- banderas para colocación y eliminación de la orden pendiente
bool pending_done=false;
bool pending_deleted=false;
//--- aquí vamos a guardar el ticket de la orden pendiente
ulong order_ticket;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- establecemos MagicNumber para marcar todas nuestras órdenes
   trade.SetExpertMagicNumber(MagicNumber);
//--- vamos a mandar las solicitude comerciales en el modo asincrónico utilizando la
   trade.SetAsyncMode(true);
//--- inicializamos la variable con un cero
   order_ticket=0;
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- colocación de orden pendiente
   if(!pending_done)
     {
      double ask=SymbolInfoDouble(_Symbol,SYMBOL_ASK);
      double buy_stop_price=NormalizeDouble(ask+1000*_Point,(int)SymbolInfoInteger(_S
      bool res=trade.BuyStop(0.1,buy_stop_price,_Symbol);
      //--- si la función BuyStop() ha trabajado con éxito
      if(res)
        {
         pending_done=true;
         //--- obtenemos el resultado del envío de la solicitud desde ctrade
         MqlTradeResult trade_result;
         trade.Result(trade_result);
         //---obtenemos request_id para la solicitud enviada
         uint request_id=trade_result.request_id;
         Print("La solicitud para colocar la orden pendiente ha sido enviada. Identif
         //--- recordamos el ticket de la orden (al usar el modo asincrónico del envío
         order_ticket=trade_result.order;
         //--- todo está hecho por eso salimos del manejador OnTick()
         return;
        }
     }
//--- eliminación de la orden pendiente
   if(!pending_deleted)
      //--- verificación adicional
      if(pending_done && (order_ticket!=0))
        {
         //--- intentamos eliminar la orden pendiente
         bool res=trade.OrderDelete(order_ticket);
         Print("OrderDelete=",res);
         //--- si la solicitud para la eliminación ha sido enviada con éxito
         if(res)
```

```
            {
             pending_deleted=true;
             //--- obtenemos el resultado de ejecución de la solicitud
             MqlTradeResult trade_result;
             trade.Result(trade_result);
             //--- extraemos del resultado el identificador de la solicitud
             uint request_id=trade_result.request_id;
             //--- mostramos en el Diario
             Print("Se ha enviado la solicitud para la eliminación de la orden pendien
                   ". Identificador de la solicitud Request_ID=",request_id,
                   "\r\n");
             //--- registramos el ticket de la orden desde el resultado de la solicitu
             order_ticket=trade_result.order;
            }
         }
//---
  }
//+------------------------------------------------------------------+
//| TradeTransaction function                                        |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
  {
//--- obtenemos el tipo de la transacción como valor de la enumeración
   ENUM_TRADE_TRANSACTION_TYPE type=(ENUM_TRADE_TRANSACTION_TYPE)trans.type;
//--- si la transacción es el resultado de procesamiento de la solicitud, mostramos s
   if(type==TRADE_TRANSACTION_REQUEST)
     {
      Print(EnumToString(type));
      //--- mostramos la descripción de la solicitud procesada
      Print("------------RequestDescription\r\n",RequestDescription(request));
      //--- mostramos la descripción del resultado de la solicitud
      Print("------------ResultDescription\r\n",TradeResultDescription(result));
      //--- recordamos el ticket de la orden para su eliminación durante el siguiente
      if(result.order!=0)
        {
         //--- eliminamos esta orden según su ticket durante la siguiente llamada de
         order_ticket=result.order;
         Print(" Ticket de la orden pendiente ",order_ticket,"\r\n");
        }
     }
   else // para la transacción de otro tipo mostramos la descripción completa
//--- mostramos la descripción de la transacción recibida en el Diario
      Print("------------TransactionDescription\r\n",TransactionDescription(trans));

//---
  }
//+------------------------------------------------------------------+
//|  Devuelve la descripción textual de la transacción               |
//+------------------------------------------------------------------+
string TransactionDescription(const MqlTradeTransaction &trans)
  {
//---
   string desc=EnumToString(trans.type)+"\r\n";
   desc+="Symbol: "+trans.symbol+"\r\n";
   desc+="Deal ticket: "+(string)trans.deal+"\r\n";
   desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
   desc+="Order ticket: "+(string)trans.order+"\r\n";
   desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
   desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
```

```
   desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
   desc+="Order expiration: "+TimeToString(trans.time_expiration)+"\r\n";
   desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
   desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
   desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
   desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
   desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
//--- devolvemos la cadena obtenida
   return desc;
  }
//+------------------------------------------------------------------+
//|  Devuelve la descripción textual de la solicitud comercial       |
//+------------------------------------------------------------------+
string RequestDescription(const MqlTradeRequest &request)
  {
//---
   string desc=EnumToString(request.action)+"\r\n";
   desc+="Symbol: "+request.symbol+"\r\n";
   desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
   desc+="Order ticket: "+(string)request.order+"\r\n";
   desc+="Order type: "+EnumToString(request.type)+"\r\n";
   desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
   desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
   desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
   desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
   desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
   desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
   desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
   desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
   desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
   desc+="Comment: "+request.comment+"\r\n";
//--- devolvemos la cadena obtenida
   return desc;
  }
//+------------------------------------------------------------------+
//|  Devuelve la descripción textual del resultado de procesamiento  |
//|  de la solicitud                                                 |
//+------------------------------------------------------------------+
string TradeResultDescription(const MqlTradeResult &result)
  {
//---
   string desc="Retcode "+(string)result.retcode+"\r\n";
   desc+="Request ID: "+StringFormat("%d",result.request_id)+"\r\n";
   desc+="Order ticket: "+(string)result.order+"\r\n";
   desc+="Deal ticket: "+(string)result.deal+"\r\n";
   desc+="Volume: "+StringFormat("%G",result.volume)+"\r\n";
   desc+="Price: "+StringFormat("%G",result.price)+"\r\n";
   desc+="Ask: "+StringFormat("%G",result.ask)+"\r\n";
   desc+="Bid: "+StringFormat("%G",result.bid)+"\r\n";
   desc+="Comment: "+result.comment+"\r\n";
//--- devolvemos la cadena obtenida
   return desc;
  }
```

Véase también

Tipos de transacciones comerciales, OnTradeTransaction()

Estructura para obtención de precios actuales (MqlTick)

Es la estructura para almacenar los últimos precios del símbolo. Sirve para recibir de una manera rápida la información más solicitada sobre los precios corrientes.

```
struct MqlTick
  {
   datetime     time;          // Hora de la última actualización de precios
   double       bid;           // Precio actual Bid
   double       ask;           // Precio actual Ask
   double       last;          // Precio actual de la última transacción (Last)
   ulong        volume;        // Volumen para el precio actual Last
  };
```

La variable del tipo MqlTick permite obtener los valores Ask, Bid, Last y Volume sólo con una llamada a la función SymbolInfoTick().

Ejemplo:

```
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());
//---
  }
```

Véase también

Estructuras y clases

Códigos de errores y advertencias

Este apartado contiene las siguientes descripciones:

- Códigos de retorno del servidor comercial – análisis de resultados del envío de una solicitud comercial mandada por la función OrderSend();
- Advertencias del compilador – códigos de los mensajes de advertencia mostrados durante la compilación (nos son errores);
- Errores de compilación – códigos de los mensajes de error en caso del intento fallido de compilación;
- Errores de tiempo de ejecución – códigos de errores durante la ejecución de un programa mql5, los que se puede obtener utilizando la función GetLastErtror().

Códigos de retorno del servidor comercial

Todas las órdenes respecto a la ejecución de las operaciones comerciales se mandan en forma de una estructura de solicitudes comerciales MqlTradeRequest a través de la función OrderSend(). El resultado de ejecución de esta función se coloca en la estructura MqlTradeResult, su campo *retcode* contiene el código de retorno del servidor comercial.

Código	Identificador	Descripción
10004	TRADE_RETCODE_REQUOTE	Recuota
10006	TRADE_RETCODE_REJECT	Solicitud rechazada
10007	TRADE_RETCODE_CANCEL	Solicitud cancelada por el agente
10008	TRADE_RETCODE_PLACED	Orden colocada
10009	TRADE_RETCODE_DONE	Solicitud ejecutada
10010	TRADE_RETCODE_DONE_PARTIAL	Solicitud ejecutada parcialmente
10011	TRADE_RETCODE_ERROR	Error al procesar la solicitud
10012	TRADE_RETCODE_TIMEOUT	Solicitud cancelada al expirar el plazo
10013	TRADE_RETCODE_INVALID	Solicitud no válida
10014	TRADE_RETCODE_INVALID_VOLUME	Volumen en la solicitud no válido
10015	TRADE_RETCODE_INVALID_PRICE	Precio en la solicitud no válido
10016	TRADE_RETCODE_INVALID_STOPS	Stops en la solicitud no válido
10017	TRADE_RETCODE_TRADE_DISABLED	Transacciones comerciales están prohibidas
10018	TRADE_RETCODE_MARKET_CLOSED	Mercado está cerrado
10019	TRADE_RETCODE_NO_MONEY	Falta de medios monetarios para cumplir la solicitud
10020	TRADE_RETCODE_PRICE_CHANGED	Precios se han cambiado
10021	TRADE_RETCODE_PRICE_OFF	Faltan las cotizaciones para procesar la solicitud
10022	TRADE_RETCODE_INVALID_EXPIRATION	Fecha de expiración no válida de la orden en la solicitud
10023	TRADE_RETCODE_ORDER_CHANGED	Estado de la orden se ha cambiado

10024	TRADE_RETCODE_TOO_MANY_REQUESTS	Solicitudes muy frecuentes
10025	TRADE_RETCODE_NO_CHANGES	Sin cambios en la solicitud
10026	TRADE_RETCODE_SERVER_DISABLES_AT	Autotrading está prohibido por el servidor
10027	TRADE_RETCODE_CLIENT_DISABLES_AT	Autotrading está prohibido por el terminal de cliente
10028	TRADE_RETCODE_LOCKED	Solicitud está bloqueada para procesar
10029	TRADE_RETCODE_FROZEN	Orden o posición están congeladas
10030	TRADE_RETCODE_INVALID_FILL	Está especificado el tipo de ejecución de orden no válido
10031	TRADE_RETCODE_CONNECTION	No hay conexión con el servidor de comercio
10032	TRADE_RETCODE_ONLY_REAL	Operación permitida únicamente para las cuentas reales
10033	TRADE_RETCODE_LIMIT_ORDERS	Alcanzado el límite del número de órdenes pendientes
10034	TRADE_RETCODE_LIMIT_VOLUME	Alcanzado el límite del volumen de órdenes y posiciones para este símbolo
10035	TRADE_RETCODE_INVALID_ORDER	Tipo de orden inválido o prohibido
10036	TRADE_RETCODE_POSITION_CLOSED	Posición con el POSITION_IDENTIFIER especificado ya está cerrada

Advertencias del compilador

Las advertencias del compilador tienen un carácter informativo, no son mensajes de error.

Número	Descripción
21	Escritura incompleta de fecha en la cadena datetime
22	Números erróneos en la cadena datetime para la fecha, se requiere: año 1970<=X<=3000 mes 0<X<=12 día 0<X<= 31/30/28(29)....
23	Números erróneos en la cadena datetime para la hora, se requiere: hora 0<=X<24 minuto 0<=X<60
24	Color incorrecto en el formato RGB: uno de los componentes RGB es menos de 0 o más de 255
25	Caracter desconocido en la secuencia de escape. Se conocen: \n \r \t \\ \" \' \X \x
26	Volumen de variables locales muy grande (>512Kb) de la función, reduzca la cantidad
29	Enumeración ya está definida (duplicación) - miembros serán añadidos a la primera definición
30	Sobrescritura de una macro
31	La variable está declarada pero no se utiliza en ningún sitio
32	Constructor tiene que ser del tipo void
33	Destructor tiene que ser del tipo void
34	Constante no entra en el rango de los enteros (X>_UI64_MAX || X<_I64_MIN) y será transformada en el tipo double
35	HEX muy largo, más de 16 caracteres significativos (se cortan los medio bytes "nibbles" mayores)
36	No hay ni un medio byte en la HEX cadena "0x"
37	No hay funciones - nada para procesar
38	Se usa una variable no inicializada
41	Función sin cuerpo, tampoco se llama

43	Posibles pérdidas de datos durante la conversión del tipo. Ejemplo: int x=(double)z;
44	Pérdida de precisión (datos) durante la conversión de una constante. Ejemplo: int x=M_PI
45	En las operaciones de comparación hay diferencia entre los signos de operandos. Ejemplo: (char)c1>(uchar)c2
46	Problemas con importación de la función - se requiere la declaración de #import o la importación de funciones ya está cerrada
47	Descripción es muy larga - los caracteres sobrantes no serán incluidos en el archivo ejecutable
48	Cantidad de buffers de indicadores declarados es menor de la requerida
49	Color para dibujar una serie gráfica en el indicador no está especificado
50	No hay series gráficas para dibujar el indicador
51	Función manejadora "OnStart" no ha sido encontrada en el script
52	Función manejadora "OnStart" está definida con parámetros erróneos
53	Función "OnStart" puede ser definida sólo en un script
54	Función "OnInit" está definida con parámetros erróneos
55	Función "OnInit" no se usa en los scripts
56	Función 'OnDeinit' está definida con parámetros erróneos
57	Función 'OnDeinit' no se usa en los scripts
58	Hay dos funciones 'OnCalculate' definidas. Se usará la función OnCalculate() en un array de precios
59	Detectado el sobrante a la hora de calcular una constante de números enteros
60	Probablemente la variable no esté inicializada.
61	Esta declaración hace que sea imposible referirse a la variable local declarada en la línea especificada

62	Esta declaración hace que sea imposible referirse a la variable global declarada en la línea especificada
63	No se puede usar para los arrays estáticos
64	Esta declaración hace imposible la referencia a la variable predefinida
65	Valor de expresión siempre es true/false
66	El uso de una variable o expresión del tipo bool en las operaciones matemáticas puede resultar inseguro
67	El resultado de aplicación del operador menos unario al tipo sin signo ulong es indefinido
68	La versión especificada en la propiedad #property version es inadmisible para la colocación en el apartado Tienda, el formato correcto sería #property version "XXX.YYY"
69	Falta de la expresión para la ejecución según la condición
70	Tipo de función que se devuelve es incorrecto, o parámetros incorrectos durante la declaración de la función-manejadora del evento
71	Se requiere la conversión de estructuras explícita al mismo tipo
72	Esta declaración hace imposible el acceso directo al miembro de una clase que ha sido declarado en esta cadena. El acceso será posible sólo con el uso de la operación de resolución de contexto ::
73	Константа в двоичной записи слишком велика, старшие разряды будут отброшены
74	Параметр в методе наследуемого класса отличается модификатором const, дочерняя функция перегрузила функцию родителя
75	Отрицательное или слишком большое значение смещения в битовой операции сдвига, результат выполнения неопределён

Errores de compilación

MetaEdtior 5 (editor de programas mql5) muestra mensajes sobre los errores del programa que han sido detectados por el compilador built-in durante el proceso de compilación. La lista de estos errores viene en la tabla de abajo. Para compilar el código fuente en un código ejecutable pulse **F7**. Los programas que tienen errores no podrán ser compilados hasta que los errores especificados por el compilador no sean corregidos.

Número	Descripción
100	Error de lectura de archivo
101	Error de apertura de un archivo *.EX5 con el fin de escribirlo para guardar
103	Memoria insuficiente para terminar la compilación
104	Unidad sintáctica vacía no reconocida por el compilador
105	Nombre del archivo incorrecto en #include
106	Error de acceso a un archivo en #include (tal vez el archivo no exista)
108	Nombre inapropiado para #define
109	Comando desconocido del preprocesador (estos valen #include,#define,#property,#import)
110	Símbolo desconocido para el compilador
111	Función sin implementar (hay descripción, pero no hay cuerpo)
112	Falta comilla doble (")
113	Falta paréntesis angular izquierdo (<) o comilla doble (")
114	Falta comilla simple (')
115	Falta paréntesis angular derecho ">"
116	En declaración no está especificado el tipo
117	No hay operador de retorno return, o hay pero no en todas las ramas de ejecución
118	Se esperaba el corchete que abre del parámetro de la llamada
119	Error de escritura EX5
120	Acceso incorrecto a los elementos de un array
121	Función no tiene el tipo void y el operador return debe devover un valor

122	Declaración del destructor incorrecta
123	Faltan dos puntos ":"
124	Variable ya está declarada
125	Variable con este identificador ya está declarada
126	Nombre de variable muy largo (>250 caracteres)
127	Estructura con este identificador ya está definida
128	Estructura no definida
129	Miembro de estructura con este nombre ya está definido
130	No existe este miembro de estructura
131	Corchetes no hacen pareja
132	Se espera el paréntesis izquierdo "("
133	Llaves sin hacer pareja (falta "}")
134	Complicado para la compilación (muchísimas ramas, la pila entera de niveles rellena)
135	Error de apertura de un archivo para la lectura
136	Falta memoria para cargar archivo de origen en la memoria
137	Se espera una variable
138	Referencia no puede ser inicializada
140	Se esperaba asignación (surge con la declaración)
141	Se espera la llave izquierda "{"
142	Sólo un array dinámico puede figurar como parámetro
143	Uso del tipo "void" es inadmisible
144	No hay par para ")" o "]", es decir, falta "(" o "["
145	No hay par para "(" o "[", es decir, falta ")" o "]"
146	Tamaño del array incorrecto
147	Demasiados parámetros (>64)
149	Este token no se espera aquí
150	Uso de operación inadmisible (operandos

	incorrectos)
151	Expresión del tipo void es inadmisible
152	Se espera operador
153	Uso incorrecto de break
154	Se espera punto y coma ";"
155	Se espera coma ","
156	Tiene que ser un tipo de clase, y no de estructura
157	Se esperaba una expresión
158	En HEX hay "un símbolo que no es HEX" o número demasiado largo (número de dígitos > 511)
159	Cadena-constante tiene más de 65534 símbolos
160	Definición de una función aquí es inaceptable
161	Fin de programa inesperado
162	Declaración adelantada está prohibida para las estructuras
163	Función con este nombre ya está definida y tiene otro tipo del valor de retorno
164	Función con este nombre ya está definida y tiene otro conjunto de parámetros
165	Función con este nombre ya está definida y implementada
166	No se ha encontrado la sobrecarga para esta función
167	Función con valor devuelto del tipo void no puede devolver valor
168	Función no definida
170	Se espera un valor
171	En la expresión case se aceptan sólo las constantes de números enteros
172	Valor para case en este switch ya ha sido usado
173	Se espera el valor de números enteros
174	En la expresión #import se espera el nombre de archivo
175	Expresiones a nivel global no están permitidas

176	Falta paréntesis ")" antes de ";"
177	A la izquierda del signo igualdad se supone una variable
178	Resultado de la expresión no se usa
179	Declaración de las variables en case no se permite
180	Conversión implícita de una cadena a un número
181	Conversión implícita de un número a una cadena
182	Llamada ambigua a una función sobrecargada (valen varias sobrecargas)
183	Inadmisible else sin correspondiente if
184	Inadmisible case o default sin correspondiente switch
185	Uso inadmisible de elipse
186	La sucesión inicializadora tiene más elementos que la variable inicializada
187	Se espera una constante para case
188	Se requiere una expresión constante
189	Una variable constante no puede ser cambiada
190	Se espera una paréntesis o coma (declaración del miembro del array)
191	Identificador de la enumeración ya se utiliza
192	Enumeración no puede tener modificadores de acceso (const, extern, static)
193	Miembro de enumeración ya ha sido declarado con otro valor
194	Existe una variable definida con el mismo nombre
195	Existe una estructura definida con el mismo nombre
196	Se espera el nombre del miembro de enumeración
197	Se espera una expresión de números enteros
198	División por cero en una expresión constante
199	Número de parámetros incorrecto en la función

200	Parámetro por referencia tiene que ser una variable
201	Se espera una variable del mismo tipo para pasarla por referencia
202	Una variable constante no puede ser pasada por referencia no constante
203	Se requiere una constante positiva de números enteros
204	Error de acceso a un miembro de clase protegido
205	Importación ya está definida por otro medio
208	Archivo ejecutable no está creado
209	Punto de entrada 'OnCalculate' para el indicador no ha sido encontrado
210	Operador continue puede ser usado sólo dentro del ciclo
211	Error de acceso al miembro de clase private (cerrado)
213	Método de estructura o clase no está declarado
214	Error de acceso al método de clase private (cerrado)
216	No se permite copiar las estructuras con objetos
218	Índice sale del rango de array
219	No se permite la inicialización de arrays en la declaración de métodos o clases
220	Constructor de clase no puede tener parámetro
221	Destructor de clase no puede tener parámetro
222	Ya está declarado método de clase o estructura con este nombre y parámetros
223	Se espera un operando
224	Ya existe método de clase o estructura con este nombre pero con otros parámetros (declaración!=implementación)
225	Función importada sin describir
227	Llamada ambigua a una función sobrecargada (coincidencia exacta de parámetros para varias sobrecargas)

228	Se espera el nombre de variable
229	No se puede declarar una referencia en este sitio
230	Ya se usa como el nombre de enumeración
232	Se espera clase o estructura
235	No se puede llamar a delete para eliminar el array
236	Se espera el operador ' while '
237	El operador delete tiene que tener un puntero
238	Ya hay default para este switch
239	Error sintáctico
240	Sucesión escape puede encontrarse sólo en las cadenas (se empieza desde '\')
241	Se requiere una matriz - corchete cuadrado '[' no pertenece a la matriz, o como parámetro-matriz ofrecen no matriz
242	No puede ser inicializado mediante la sucesión inicializadora
243	Importación no definida
244	Error del optimizador en el árbol sintáctico
245	Han sido declaradas demasiadas estructuras (simplifique el programa)
246	Conversión del parámetro no está permitida
247	Uso incorrecto del operador delete
248	No se puede declarar un puntero a una referencia
249	No se puede declarar una referencia a una referencia
250	No se puede declarar un puntero a un puntero
251	Declaración de estructura en la lista de parámetros es inadmisible
252	Operación de conversión de tipos inadmisible
253	Un puntero puede ser declarado sólo para una clase o estructura
256	Identificador no declarado
257	Error del optimizador del código ejecutable

258	Error de generación del código ejecutable
260	Expresión inadmisible para el operador switch
261	Pool de constantes de cadenas está sobrelleno, simplifique el programa
262	Imposible convertir a una enumeración
263	No se puede usar virtual para los datos (miembros de clase o estructura)
264	No se puede llamar al método de clase protegido
265	Función virtual sobrescrita devuelve otro tipo
266	No se puede heredar una clase de una estructura
267	No se puede heredar una estructura de una clase
268	Constructor no puede ser virtual (especificador virtual es inadmisible)
269	Estructura no puede tener métodos virtuales
270	Una función debe tener curpo
271	Sobrecarga de funciones de sistema (funciones del terminal) está prohibida
272	Especificador const está prohibido para las funciones que no son miembros de una clase o estructura
274	No se puede cambiar miembros de clase en el método constante
276	Sucesión inicializadora inapropiada
277	Falta valor por defecto para el parámetro (particularidad de declaración de parámetros por defecto)
278	Sobrescritura de parámetro por defecto (diferentes valores en declaración y implementación)
279	No se puede llamar a método no constante para un objeto constante
280	Para acceder a los miembros se necesita un objeto (un punto está puesto para no clase/ estructura)
281	No se puede usar el nombre de una estructura ya declarada para declaración

284	Conversión no permitida (con la herencia cerrada)
285	Estructuras y arrays no pueden ser usados como variables input
286	Especificador const es inadmisible para un constructor/destructor
287	Expresión de cadena incorrecto para el tipo datetime
288	Propiedad desconocida (#property)
289	Valor incorrecto para la propiedad
290	Índice incorrecto para la propiedad #property
291	Parámetro de llamada ha sido omitido - < func (x,) >
293	Objeto tiene que ser pasado por referencia
294	Array tiene que ser pasado por referencia
295	Función ha sido declarada como importada
296	Función ha sido declarada como exportada
297	No se puede exportar una función importada
298	Función importada no puede tener este parámetro (no se puede pasar puntero, clase o estructura que contiene un array dinámico, puntero, clase, etc.)
299	Tiene que ser una clase
300	Sección #import no está cerrada
302	Falta de correspondencia de tipos
303	extern-variable ya está inicializada
304	No se ha encontrado ni una función exportada o punto estándar de entrada
305	Prohibida la llamada explícita del constructor
306	El método ha sido declarado como método constante
307	El método no ha sido declarado como método constante
308	Tamaño incorrecto del archivo de recurso
309	Nombre del recurso incorrecto
310	Error de apertura del archivo de recurso

311	Error de lectura del archivo de recurso
312	Tipo de recurso desconocido
313	Ruta incorrecta hacia el archivo del recurso
314	El nombre especificado del recurso ya se utiliza
315	Se esperaban los parámetros de la macro
316	Tras el nombre de la macro debe ir el espacio
317	Error en la descripción de los parámetros de la macro
318	Número de parámetros inválido a la hora de usar la macro
319	Número máximo de parámetros (16) para la macro ha sido superado
320	La macro es muy complicada, hay que hacerla más simple
321	Sólo una enumeración puede ser el parámetro EnumToString()
322	Nombre del recurso es demasiado largo
323	Formato de imagen insoportable (se admite sólo el formato BMP con la profundidad del color de 24 o 32 bits)
324	Prohibido declarar un array dentro del operador
325	La función se puede definir sólo al nivel global
326	Esta declaración es inaceptable para el área de visibilidad actual (zona de declaración)
327	La inicialización de las variables estáticas con valores locales no está permitida
328	La declaración de un array de objetos que no disponen del constructor por defecto no está permitida
329	La lista de inicialización está permitida únicamente para los constructores
330	Falta la definición de la función después de la lista de inicialización
331	La lista de inicialización está vacía
332	La inicialización de un array en el constructor está prohibida
333	En la lista de inicialización está prohibido inicializar los miembros de la clase base

334	Se esperaba la expresión del tipo entero
335	El volumen de memoria requerido para el array supera el valor máximo permitido
336	El volumen de memoria requerido para la estructura supera el valor máximo permitido
337	El volumen de memoria requerido para las variables declaradas a nivel global supera el valor máximo permitido
338	El volumen de memoria requerido para las variables locales supera el valor máximo permitido
339	Constructor no definido
340	Nombre inválido para el archivo del icono
341	No se ha podido abrir el archivo del icono según la ruta especificada
342	El archivo del icono es incorrecto y no corresponde al formato ICO
343	La reinicialización del miembro en el constructor de la clase/estructura utilizando la lista de inicialización
344	La inicialización de los miembros estáticos en la lista de inicialización del constructor no está permitida
345	La inicialización del miembro no estático de la estructura/clase está prohibida a nivel global
346	El nombre del método de la clase/estructura coincide con el nombre del miembro declarado antes
347	El nombre del miembro de la clase/estructura coincide con el nombre del método declarado antes
348	Una función virtual no puede ser declarada como static
349	El modificador const no está permitido para una función estática
350	Un constructor o destructor no pueden ser estáticos
351	No se puede acceder a un miembro/método no estático de la clase o estructura desde una función estática

352	Tras la palabra clave operator se espera una operación de sobrecarga (+,-,[],++,-- etc.)
353	No todas las operaciones se puede sobrecargar en MQL5
354	La definición no corresponde a la declaración
355	Número de parámetros para el operador incorrecto
356	No se ha encontrado ninguna función del manejador de evento
357	No se puede exportar los métodos
358	No se puede normalizar el puntero a un objeto constante con el puntero a un objeto no constante
359	De momento las plantillas de las clases no se soportan
360	Sobrecarga de las plantillas de las funciones de momento no se soporta
361	Imposible aplicar la plantilla de la función
362	Parámetro ambiguo en la plantilla de la función (se encajan varios tipos de parámetros)
363	Imposible determinar con qué tipo de parámetro normalizar el argumento de la plantilla de la función
364	Número incorrecto de parámetros en la plantilla de la función
365	Plantilla de funciones no puede ser virtual
366	Plantillas de funciones no pueden ser exportadas
367	No se puede importar las plantillas de funciones
368	Estructuras que contienen objetos no están permitidas

Errores de tiempo de ejecución

GetLastError() es la función que devuelve el código del último error que se almacena en la variable predefinida _LastError. El valor de esta variable puede ser puesto a cero usando la función ResetLastError().

Constante	Valor	Descripción
ERR_INTERNAL_ERROR	4001	Error interno inesperado
ERR_WRONG_INTERNAL_PARAMETER	4002	Parámetro erróneo durante la llamada built-in a la función del terminal de cliente
ERR_INVALID_PARAMETER	4003	Parámetro erróneo durante la llamada a la función de sistema
ERR_NOT_ENOUGH_MEMORY	4004	No hay memoria suficiente para ejecutar la función de sistema
ERR_STRUCT_WITHOBJECTS_ORCLASS	4005	Estructura contiene objetos de cadenas y/o de arrays dinámicos y/o estructuras con estos objetos y/o clases
ERR_INVALID_ARRAY	4006	Array del tipo inapropiado, tamaño inapropiado o objeto dañado del array dinámico
ERR_ARRAY_RESIZE_ERROR	4007	No hay memoria suficiente para la reubicación de un array, o un intento de cambio del tamaño de un array estático
ERR_STRING_RESIZE_ERROR	4008	No hay memoria suficiente para la reubicación de una cadena
ERR_NOTINITIALIZED_STRING	4009	Cadena no inicializada
ERR_INVALID_DATETIME	4010	Valor de fecha y/o hora incorrecto
ERR_ARRAY_BAD_SIZE	4011	Tamaño solicitado del array supera 2 gigabytes
ERR_INVALID_POINTER	4012	Puntero erróneo
ERR_INVALID_POINTER_TYPE	4013	Tipo erróneo del puntero
ERR_FUNCTION_NOT_ALLOWED	4014	Función de sistema no está permitida para la llamada
ERR_RESOURCE_NAME_DUPLIC	4015	Coincidencia del nombre del

ATED		recurso dinámico y estático
ERR_RESOURCE_NOT_FOUND	4016	Recurso con este nombre no encontrado en EX5
ERR_RESOURCE_UNSUPPOTED_TYPE	4017	Tipo del recurso no soportado o el tamaño superior a 16 Mb
ERR_RESOURCE_NAME_IS_TOO_LONG	4018	Nombre del recurso supera 63 caracteres
Gráficos		
ERR_CHART_WRONG_ID	4101	Identificador erróneo del gráfico
ERR_CHART_NO_REPLY	4102	Gráfico no responde
ERR_CHART_NOT_FOUND	4103	Gráfico no encontrado
ERR_CHART_NO_EXPERT	4104	Gráfico no tiene un Asesor Experto que pueda procesar el evento
ERR_CHART_CANNOT_OPEN	4105	Error al abrir el gráfico
ERR_CHART_CANNOT_CHANGE	4106	Error al cambiar el símbolo y período del gráfico
ERR_CHART_WRONG_PARAMETER	4107	Valor erróneo del parámetro para la función de trabajo con los gráficos
ERR_CHART_CANNOT_CREATE_TIMER	4108	Error al crear el temporizador
ERR_CHART_WRONG_PROPERTY	4109	Identificador erróneo de la propiedad del gráfico
ERR_CHART_SCREENSHOT_FAILED	4110	Error al crear un screenshot
ERR_CHART_NAVIGATE_FAILED	4111	error de navegación por el gráfico
ERR_CHART_TEMPLATE_FAILED	4112	Error al aplicar una plantilla
ERR_CHART_WINDOW_NOT_FOUND	4113	Subventana que contiene el indicador especificado no encontrada
ERR_CHART_INDICATOR_CANNOT_ADD	4114	Error al insertar un indicador en el gráfico
ERR_CHART_INDICATOR_CANNOT_DEL	4115	Error al quitar un indicador desde el gráfico
ERR_CHART_INDICATOR_NOT	4116	El indicador no ha sido

_FOUND		encontrado en el gráfico especificado
Objetos gráficos		
ERR_OBJECT_ERROR	4201	Error al manejar un objeto gráfico
ERR_OBJECT_NOT_FOUND	4202	Objeto gráfico no encontrado
ERR_OBJECT_WRONG_PROPERTY	4203	Identificador erróneo de la propiedad del objeto gráfico
ERR_OBJECT_GETDATE_FAILED	4204	Imposible recibir fecha correspondiente al valor
ERR_OBJECT_GETVALUE_FAILED	4205	Imposible recibir valor correspondiente a la fecha
MarketInfo		
ERR_MARKET_UNKNOWN_SYMBOL	4301	Símbolo desconocido
ERR_MARKET_NOT_SELECTED	4302	Símbolo no está seleccionado en MarketWatch
ERR_MARKET_WRONG_PROPERTY	4303	Identificador erróneo de la propiedad del símbolo
ERR_MARKET_LASTTIME_UNKNOWN	4304	Hora del último tick no se conoce (no había ticks)
ERR_MARKET_SELECT_ERROR	4305	Error al agregar o eliminar el símbolo a/de MarketWatch
Acceso al historial		
ERR_HISTORY_NOT_FOUND	4401	Historial solicitado no encontrado
ERR_HISTORY_WRONG_PROPERTY	4402	Identificador erróneo de la propiedad del historial
Global_Variables		
ERR_GLOBALVARIABLE_NOT_FOUND	4501	Variable global del terminal de cliente no encontrada
ERR_GLOBALVARIABLE_EXISTS	4502	Variable global del terminal de cliente con este nombre ya existe
ERR_MAIL_SEND_FAILED	4510	Envío de carta fallido
ERR_PLAY_SOUND_FAILED	4511	Reproducción de sonido fallido
ERR_MQL5_WRONG_PROPERTY	4512	Identificador erróneo de la propiedad del programa

ERR_TERMINAL_WRONG_PROPERTY	4513	Identificador erróneo de la propiedad del terminal
ERR_FTP_SEND_FAILED	4514	Envío de archivo a través de ftp fallido
ERR_NOTIFICATION_SEND_FAILED	4515	No se ha podido enviar la notificación
ERR_NOTIFICATION_WRONG_PARAMETER	4516	Parámetro incorrecto para el envío de la notificación - en la función SendNotification() han pasado una línea vacía o NULL
ERR_NOTIFICATION_WRONG_SETTINGS	4517	Ajustes incorrectos de las notificaciones en el terminal (ID no especificada o permiso no concedido)
ERR_NOTIFICATION_TOO_FREQUENT	4518	Envío de notificaciones muy frecuente
Buffers de indicadores personalizados		
ERR_BUFFERS_NO_MEMORY	4601	No hay memoria suficiente para la redistribución de buffers de indicadores
ERR_BUFFERS_WRONG_INDEX	4602	Índice erróneo de su búfer de indicadores
Propiedades de indicadores personalizados		
ERR_CUSTOM_WRONG_PROPERTY	4603	Identificador erróneo de la propiedad del indicador personalizado
Account		
ERR_ACCOUNT_WRONG_PROPERTY	4701	Identificador erróneo de la propiedad de la cuenta
ERR_TRADE_WRONG_PROPERTY	4751	Identificador erróneo de la propiedad de la actividad comercial
ERR_TRADE_DISABLED	4752	Prohibida la actividad comercial para el Asesor Experto
ERR_TRADE_POSITION_NOT_FOUND	4753	Posición no encontrada
ERR_TRADE_ORDER_NOT_FOUND	4754	Orden no encontrada

ERR_TRADE_DEAL_NOT_FOUND	4755	Transacción no encontrada
ERR_TRADE_SEND_FAILED	4756	Envío de solicitud comercial fallida
Indicadores		
ERR_INDICATOR_UNKNOWN_SYMBOL	4801	Símbolo desconocido
ERR_INDICATOR_CANNOT_CREATE	4802	No se puede crear indicador
ERR_INDICATOR_NO_MEMORY	4803	Memoria insuficiente para añadir el indicador
ERR_INDICATOR_CANNOT_APPLY	4804	Indicador no puede ser aplicado a otro indicador
ERR_INDICATOR_CANNOT_ADD	4805	Error al añadir indicador
ERR_INDICATOR_DATA_NOT_FOUND	4806	Datos solicitados no encontrados
ERR_INDICATOR_WRONG_HANDLE	4807	Handle del indicador es erróneo
ERR_INDICATOR_WRONG_PARAMETERS	4808	Número erróneo de parámetros al crear un indicador
ERR_INDICATOR_PARAMETERS_MISSING	4809	No hay parámetros cuando se crea un indicador
ERR_INDICATOR_CUSTOM_NAME	4810	El primer parámetro en la matriz tiene que ser el nombre del indicador personalizado
ERR_INDICATOR_PARAMETER_TYPE	4811	Tipo erróneo del parámetro en la matriz al crear un indicador
ERR_INDICATOR_WRONG_INDEX	4812	Índice del búfer de indicador que se solicita es erróneo
Profundidad de mercado		
ERR_BOOKS_CANNOT_ADD	4901	No se puede añadir la profundidad de mercado
ERR_BOOKS_CANNOT_DELETE	4902	No se puede eliminar la profundidad de mercado
ERR_BOOKS_CANNOT_GET	4903	No se puede obtener los datos de la profundidad de mercado
ERR_BOOKS_CANNOT_SUBSCRIBE	4904	Error al suscribirse a la recepción de nuevos datos de

		la profundidad de mercado
Operaciones con archivos		
ERR_TOO_MANY_FILES	5001	No se puede abrir más de 64 archivos
ERR_WRONG_FILENAME	5002	Nombre del archivo no válido
ERR_TOO_LONG_FILENAME	5003	Nombre del archivo demasiado largo
ERR_CANNOT_OPEN_FILE	5004	Error al abrir el archivo
ERR_FILE_CACHEBUFFER_ERROR	5005	Memoria insuficiente para la caché de lectura
ERR_CANNOT_DELETE_FILE	5006	Error al eliminar el archivo
ERR_INVALID_FILEHANDLE	5007	Archivo con este manejador ya está cerrado, o no se abría en absoluto
ERR_WRONG_FILEHANDLE	5008	Manejador erróneo de archivo
ERR_FILE_NOTTOWRITE	5009	El archivo debe ser abierto para la escritura
ERR_FILE_NOTTOREAD	5010	El archivo debe ser abierto para la lectura
ERR_FILE_NOTBIN	5011	El archivo debe ser abierto como un archivo binario
ERR_FILE_NOTTXT	5012	El archivo debe ser abierto como un archivo de texto
ERR_FILE_NOTTXTORCSV	5013	El archivo debe ser abierto como un archivo de texto o CSV
ERR_FILE_NOTCSV	5014	El archivo debe ser abierto como un archivo CSV
ERR_FILE_READERROR	5015	Error de lectura de archivo
ERR_FILE_BINSTRINGSIZE	5016	Hay que especificar el tamaño de la cadena porque el archivo ha sido abierto como binario
ERR_INCOMPATIBLE_FILE	5017	Para los arrays de cadenas - un archivo de texto, para los demás - un archivo binario
ERR_FILE_IS_DIRECTORY	5018	No es un archivo, es un directorio
ERR_FILE_NOT_EXIST	5019	Archivo no existe
ERR_FILE_CANNOT_REWRITE	5020	No se puede reescribir el

		archivo
ERR_WRONG_DIRECTORYNAME	5021	Nombre erróneo del directorio
ERR_DIRECTORY_NOT_EXIST	5022	Directorio no existe
ERR_FILE_ISNOT_DIRECTORY	5023	Es un archivo, no es un directorio
ERR_CANNOT_DELETE_DIRECTORY	5024	No se puede eliminar el directorio
ERR_CANNOT_CLEAN_DIRECTORY	5025	No se puede limpiar el directorio (tal vez, uno o más archivos estén bloqueados y no se ha podido llevar a cabo la eliminación)
ERR_MQL_FILE_WRITEERROR	5026	No se ha podido escribir el recurso en el archivo
Conversión de cadenas		
ERR_NO_STRING_DATE	5030	No hay fecha en la cadena
ERR_WRONG_STRING_DATE	5031	Fecha errónea en la cadena
ERR_WRONG_STRING_TIME	5032	Hora errónea en la cadena
ERR_STRING_TIME_ERROR	5033	Error de conversión de cadena a fecha
ERR_STRING_OUT_OF_MEMORY	5034	Memoria insuficiente para la cadena
ERR_STRING_SMALL_LEN	5035	Longitud de cadena es menos de la esperada
ERR_STRING_TOO_BIGNUMBER	5036	Número excesivamente grande, más que ULONG_MAX
ERR_WRONG_FORMATSTRING	5037	Cadena de formato errónea
ERR_TOO_MANY_FORMATTERS	5038	Hay más especificadores de formato que los parámetros
ERR_TOO_MANY_PARAMETERS	5039	Hay más Parámetros que los especificadores de formato
ERR_WRONG_STRING_PARAMETER	5040	Parámetro del tipo string dañado
ERR_STRINGPOS_OUTOFRANGE	5041	Posición fuera de los límites de la cadena
ERR_STRING_ZEROADDED	5042	Al final de la cadena se ha añadido 0, una operación inútil

ERR_STRING_UNKNOWNTYPE	5043	Tipo de datos desconocido durante la conversión a una cadena
ERR_WRONG_STRING_OBJECT	5044	Objeto de cadena dañado
Operaciones con matrices		
ERR_INCOMPATIBLE_ARRAYS	5050	Copiado de los arrays incompatibles. Un array de cadena puede ser copiado sólo en un array de cadena, un array numérico sólo en un array numérico
ERR_SMALL_ASSERIES_ARRAY	5051	El array que recibe está declarado como AS_SERIES, y no tiene el tamaño suficiente
ERR_SMALL_ARRAY	5052	Un array muy pequeño, posición de inicio está fuera de los límites del array
ERR_ZEROSIZE_ARRAY	5053	Un array de longitud cero
ERR_NUMBER_ARRAYS_ONLY	5054	Tiene que ser un array numérico
ERR_ONEDIM_ARRAYS_ONLY	5055	Tiene que ser un array unidimensional
ERR_SERIES_ARRAY	5056	No se puede usar serie temporal
ERR_DOUBLE_ARRAY_ONLY	5057	Tiene que ser un array del tipo double
ERR_FLOAT_ARRAY_ONLY	5058	Tiene que ser un array del tipo float
ERR_LONG_ARRAY_ONLY	5059	Tiene que ser un array del tipo long
ERR_INT_ARRAY_ONLY	5060	Tiene que ser un array del tipo int
ERR_SHORT_ARRAY_ONLY	5061	Tiene que ser un array del tipo short
ERR_CHAR_ARRAY_ONLY	5062	Tiene que ser un array del tipo char
Trabajo con OpenCL		
ERR_OPENCL_NOT_SUPPORTED	5100	Las funciones OpenCL no se soportan en este ordenador
ERR_OPENCL_INTERNAL	5101	Error interno al ejecutar

		OpenCL
ERR_OPENCL_INVALID_HANDLE	5102	Manejado OpenCL incorrecto
ERR_OPENCL_CONTEXT_CREATE	5103	Error al crear el contexto OpenCL
ERR_OPENCL_QUEUE_CREATE	5104	Error al crear la cola de ejecución en OpenCL
ERR_OPENCL_PROGRAM_CREATE	5105	Error al compilar el programa OpenCL
ERR_OPENCL_TOO_LONG_KERNEL_NAME	5106	Punto de entrada demasiado largo (kernel OpenCL)
ERR_OPENCL_KERNEL_CREATE	5107	Error al crear el kernel - punto de entrada de OpenCL
ERR_OPENCL_SET_KERNEL_PARAMETER	5108	Error al establecer los parámetros para el kernel OpenCL (punto de entrada en el programa OpenCL)
ERR_OPENCL_EXECUTE	5109	Error de ejecución del programa OpenCL
ERR_OPENCL_WRONG_BUFFER_SIZE	5110	Tamaño del búfer OpenCL incorrecto
ERR_OPENCL_WRONG_BUFFER_OFFSET	5111	Desplazamiento incorrecto en el búfer OpenCL
ERR_OPENCL_BUFFER_CREATE	5112	Error de creación del búfer OpenCL
Errores de usuario		
ERR_USER_ERROR_FIRST	65536	A partir de este código se empiezan los errores definidos por el usuario

Constantes de entrada/salida

Constantes:

- Banderas de apertura de archivos
- Propiedades de archivos
- Posicionamiento dentro del archivo
- Uso de página de código
- MessageBox

Banderas de apertura de archivos

Estos son los valores de banderas que determinan el modo de manejar un archivo. Las banderas están determinadas como sigue:

Identificador	Valor	Descripción
FILE_READ	1	Archivo se abre para la lectura. La bandera se usa cuando un archivo se abre, en la función (FileOpen()). Cuando abrimos un archivo, es obligatorio indicar la bandera FILE_WRITE y/o la bandera FILE_READ.
FILE_WRITE	2	Archivo se abre para la escritura. La bandera se usa cuando un archivo se abre, en la función (FileOpen()). Cuando abrimos un archivo, es obligatorio indicar la bandera FILE_WRITE y/o la bandera FILE_READ.
FILE_BIN	4	Modo binario de lectura-escritura (sin conversión de una cadena y en una cadena). La bandera se usa cuando un archivo se abre, en la función (FileOpen())
FILE_CSV	8	Archivo del tipo csv (todos sus elementos se convierten a las cadenas del tipo correspondiente, unicode o ansi, y se separan con un delimitador). La bandera se usa cuando un archivo se abre, en la función (FileOpen())
FILE_TXT	16	Archivo de texto simple (igual que el archivo csv pero sin tomar en cuenta los delimitadores). La bandera se usa cuando un archivo se abre, en la función (FileOpen())
FILE_ANSI	32	Cadenas del tipo ANSI (símbolos de un byte). La bandera se usa cuando un archivo se abre, en la función (FileOpen())
FILE_UNICODE	64	Cadenas del tipo UNICODE (símbolos de dos byte). La

		bandera se usa cuando un archivo se abre, en la función (FileOpen())
FILE_SHARE_READ	128	Acceso compartido para la lectura desde varios programas. La bandera se usa cuando se abre un archivo (FileOpen()), pero durante la apertura no sustituye la necesidad de indicar FILE_WRITE y/o la bandera FILE_READ
FILE_SHARE_WRITE	256	Acceso compartido para la escritura desde varios programas. La bandera se usa cuando se abre un archivo (FileOpen()), pero durante la apertura no sustituye la necesidad de indicar FILE_WRITE y/o la bandera FILE_READ
FILE_REWRITE	512	Posibilidad de reescribir un archivo usando las funciones FileCopy() y FileMove(). El archivo tiene que existir o abrirse para la escritura. En caso contrario el archivo no se abrirá.
FILE_COMMON	4096	Ubicación del archivo en la carpeta general de todos los terminales de cliente. La bandera se usa en las funciones (FileOpen()), (FileCopy(), FileMove()), (FileIsExist())

Durante la apertura de un archivo se puede indicar una o más banderas. Esto se llama la combinación de banderas. Esta combinación se escribe mediante el símbolo del lógico OR (|), éste se coloca entre las banderas especificadas. Por ejemplo, para abrir un archivo en el formato CSV para la lectura y escritura al mismo tiempo, podemos indicar la combinación FILE_READ|FILE_WRITE|FILE_CSV.

Ejemplo:

```
int filehandle=FileOpen(filename,FILE_READ|FILE_WRITE|FILE_CSV);
```

Existen unas particularidades a la hora de indicar las banderas de lectura y escritura:

- Si indicamos FILE_READ, se intenta abrir un archivo que ya existe. Si este archivo no existe, no se puede abrirlo y el nuevo no se crea.
- Si ponemos FILE_READ|FILE_WRITE, se crea un archivo nuevo, si el archivo con el nombre especificado no existe.

- FILE_WRITE – el archivo vuelve a crearse teniendo el tamaño cero.

Cuando abrimos un archivo, es obligatorio indicar la bandera FILE_WRITE y/o la bandera FILE_READ.

Las banderas que determinan el tipo de lectura de un archivo abierto poseen la prioridad. La mayor prioridad tiene la bandera FILE_CSV, luego FILE_BIN, y luego FILE_TXT que tiene la menor prioridad. De esta manera, si están indicadas varias banderas (FILE_TXT|FILE_CSV o FILE_TXT|FILE_BIN o FILE_BIN|FILE_CSV), se usará la bandera con la mayor prioridad.

Las bandera que determinan el tipo de codificación también tienen la prioridad. La bandera FILE_UNICODE es de mayor prioridad que la bandera FILE_ANSI. Por tanto, al indicar la combinación FILE_UNICODE|FILE_ANSI, se usa la bandera FILE_UNICODE.

Si no se indica FILE_UNICODE, ni tampoco FILE_ANSI, se sobreentiende FILE_UNICODE. Si no se indica FILE_CSV, ni FILE_BIN, ni FILE_TXT, entonces se sobreentiende FILE_CSV.

Si un archivo está abierto para la lectura como un archivo de texto (FILE_TXT o FILE_CSV), y además, al principio de este archivo se encuentra `0xff,0xfe` de dos bytes, la bandera de codificación será FILE_UNICODE, incluso si se especifica la bandera FILE_ANSI.

Véase también

Operaciones con archivos

Propiedades de archivos

Para obtener las propiedades de archivos se utiliza la función FileGetInteger(). Durante la llamada se le pasa el identificador de la propiedad requerida desde la enumeración ENUM_FILE_PROPERTY_INTEGER

ENUM_FILE_PROPERTY_INTEGER

Identificador	Descripción del identificador
FILE_EXISTS	Comprobación de existencia
FILE_CREATE_DATE	Fecha de creación
FILE_MODIFY_DATE	Fecha de última modificación
FILE_ACCESS_DATE	Fecha del último acceso al archivo
FILE_SIZE	Tamaño del archivo en bytes
FILE_POSITION	Posición del puntero en el archivo
FILE_END	Obtención del signo del final del archivo
FILE_LINE_END	Obtención del signo del final de la línea
FILE_IS_COMMON	El archivo está abierto en la carpeta común de todos los terminales de cliente (ver FILE_COMMON)
FILE_IS_TEXT	El archivo está abierto como un archivo de texto (ver FILE_TXT)
FILE_IS_BINARY	El archivo está abierto como un archivo binario (ver FILE_BIN)
FILE_IS_CSV	El archivo está abierto como un archivo CSV (ver FILE_CSV)
FILE_IS_ANSI	El archivo está abierto como un archivo ANSI (ver FILE_ANSI)
FILE_IS_READABLE	El archivo está abierto con posibilidades de lectura (ver FILE_READ)
FILE_IS_WRITABLE	El archivo está abierto con posibilidades de escritura (ver FILE_WRITE)

La función FileGetInteger() tiene dos diferentes opciones de llamada. En la primera, para obtener las propiedades del archivo, se indica su manejador obtenido durante la apertura del archivo por medio de la función FileOpen(). Esta variante permite obtener todas las propiedades del archivo.

La segunda opción de la función FileGetInteger() devuelve los valores de las propiedades del archivo por su nombre. Esta opción permite obtener sólo las siguientes propiedades comunes:

- FILE_EXISTS – existencia del archivo con el nombre especificado;
- FILE_CREATE_DATE – fecha de creación del archivo con el nombre especificado;

- FILE_MODIFY_DATE – fecha de modificación del archivo con el nombre especificado;
- FILE_ACCESS_DATE – fecha del último acceso al archivo con el nombre especificado;
- FILE_SIZE – tamaño del archivo con el nombre especificado.

Si intenta obtener otras propiedades aparte de las arriba mencionadas, la segunda variante de la llamada a la función FileGetInteger() devolverá un error.

Posicionamiento dentro del archivo

La mayor parte de las funciones de archivos está vinculada con las operaciones de lectura/escritura de información. Usando la función FileSeek() podemos especificar la posición de un puntero de archivos sobre una posición dentro del archivo a partir de la cual va a realizarse la siguiente operación de lectura o escritura. La enumeración ENUM_FILE_POSITION contiene las posiciones válidas de un puntero respecto al que podemos especificar el desplazamiento en bytes para la operación siguiente.

ENUM_FILE_POSITION

Identificador	Descripción
SEEK_SET	Principio del archivo
SEEK_CUR	Posición actual de un puntero de archivos
SEEK_END	Final del archivo

Véase también

FileIsEnding, FileIsLineEnding

Uso de la página de código en las operaciones de conversión de cadenas

En el lenguaje MQL5, durante las operaciones de conversión de variables de cadenas a los arrays del tipo char y viceversa, se utiliza la codificación que por defecto corresponde a la actual codificación ANSI del sistema operativo Windows (CP_ACP). Si hace falta especificar otro tipo de codificación, se puede definirlo usando un parámetro adicional para las funciones CharArrayToString(), StringToCharArray() y FileOpen().

En la tabla de abajo vienen las constantes built-in para algunas de las páginas de códigos más usadas. Las páginas de códigos que no vienen en esta tabla pueden ser definidas con el código correspondiente a esta página.

Constantes built-in de las páginas de códigos

Constante	Valor	Descripción
CP_ACP	0	Página de código actual de codificación ANSI en el sistema operativo Windows
CP_OEMCP	1	Página de código actual OEM.
CP_MACCP	2	Página de código actual Macintosh. **Nota:** Este valor suelen usar en los códigos de programas creados anteriormente, y actualmente no hay necesidad de usarlo porque los modernos ordenadores Macintosh utilizan la codificación Unicode.
CP_THREAD_ACP	3	Codificación Windows ANSI para el hilo de ejecución corriente.
CP_SYMBOL	42	Página de código Symbol
CP_UTF7	65000	Página de código UTF-7.
CP_UTF8	65001	Página de código UTF-8.

Constantes de la ventana de diálogo MessageBox

Códigos de retorno de la función MessageBox(). Si la ventana de mensaje dispone del botón Cancelar (Cancel), la función devuelve el valor IDCANCEL al apretar la tecla ESC o al pulsar el botón Cancelar (Cancel). Si la ventana de mensaje no dispone del botón Cancelar (Cancel), pulsando ESC no provoca efecto alguno.

Constante	Valor	Descripción
IDOK	1	El botón "OK" está seleccionado
IDCANCEL	2	El botón "Cancelar" (Cancel) está seleccionado
IDABORT	3	El botón "Interrumpir" (Abort) está seleccionado
IDRETRY	4	El botón "Reintentar" (Retry) está seleccionado
IDIGNORE	5	El botón "Ignorar" (Ignore) está seleccionado
IDYES	6	El botón "Sí" (Yes) está seleccionado
IDNO	7	El botón "No" (No) está seleccionado
IDTRYAGAIN	10	El botón "Repetir" (Try Again) está seleccionado
IDCONTINUE	11	El botón "Continuar" (Continue) está seleccionado

Las banderas principales de la función MessageBox() definen el contenido y comportamiento de la ventana de diálogo. Este valor puede ser una combinación de los siguientes grupos de banderas:

Constante	Valor	Descripción
MB_OK	0x00000000	La ventana de diálogo contiene un botón: OK. Por defecto
MB_OKCANCEL	0x00000001	La ventana de diálogo contiene dos botones: OK y Cancel
MB_ABORTRETRYIGNORE	0x00000002	La ventana de diálogo contiene tres botones: Abort, Retry y Ignore

MB_YESNOCANCEL	0x00000003	La ventana de diálogo contiene tres botones: Yes, No y Cancel
MB_YESNO	0x00000004	La ventana de diálogo contiene dos botones: Yes y No
MB_RETRYCANCEL	0x00000005	La ventana de diálogo contiene dos botones: Retry y Cancel
MB_CANCELTRYCONTINUE	0x00000006	La ventana de diálogo contiene tres botones: Cancel, Try Again, Continue

Para mostrar un ícono en la ventana de diálogo es necesario especificar las banderas adicionales:

Constante	Valor	Descripción
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	0x00000010	El ícono del signo STOP
MB_ICONQUESTION	0x00000020	El ícono del signo interrogación
MB_ICONEXCLAMATION, MB_ICONWARNING	0x00000030	El ícono del signo de admiración
MB_ICONINFORMATION, MB_ICONASTERISK	0x00000040	El ícono del signo **i** dentro del círculo

Los botones predefinidos se definen mediante las siguientes banderas:

Constante	Valor	Descripción
MB_DEFBUTTON1	0x00000000	El primer botón MB_DEFBUTTON1 - el botón está seleccionado por defecto, si MB_DEFBUTTON2, MB_DEFBUTTON3, o MB_DEFBUTTON4 no están especificados
MB_DEFBUTTON2	0x00000100	El segundo botón - botón por defecto
MB_DEFBUTTON3	0x00000200	El tercer botón - botón por defecto
MB_DEFBUTTON4	0x00000300	El cuarto botón - botón por defecto

Programas MQL5

Para que el programa mql5 pueda funcionar, tiene que estar compilado (botón "Compilar" o tecla F7). La compilación debe pasar sin errores(se admiten advertencias que hace falta analizar). Durante este proceso hay que crear un archivo ejecutable con el mismo nombre y la extensión EX5 en el directorio correspondiente, *terminal_dir\MQL5\Experts*, *terminal_dir\MQL5\indicators* o *terminal_dir\MQL5\scripts*. Precisamente este archivo puede ser ejecutado.

Las particularidades de funcionamiento de los programas mql5 se describen en los siguientes apartados:

- Ejecución de programas – orden de llamada a las funciones-manejadores de eventos predefinidas;
- Prueba de estrategias de trading – particularidades de funcionamiento de programas en el Probador de Estrategias;
- Eventos de terminal de cliente – descripción de eventos que pueden ser procesados en los programas;
- Llamada a las funciones importadas – orden de descripción, parámetros admisibles, orden de búsqueda y acuerdos de enlaces de las funciones importadas;
- Errores de ejecución – obtención de información sobre los errores de ejecución y errores críticos.

Los Asesores Expertos, indicadores personalizados y scripts se adjuntan a uno de los gráficos abiertos arrastrándolos con el ratón desde la ventana "Navegador" del terminal de cliente hasta el gráfico correspondiente (tecnología Drag'n'Drop). Los programas mql5 pueden trabajar sólo con el terminal de cliente que esté en funcionamiento.

Para que el Asesor Experto deje de trabajar, hay que eliminarlo del gráfico seleccionando "Asesores Expertos - Eliminar" del menú contextual de gráfico. Además, el estado del botón "Activar/desactivar Asesores Expertos" también influye en el funcionamiento de Asesor.

Para que el indicador de usuario deje de trabajar, hay que eliminarlo del gráfico.

Los indicadores personalizados y Asesores Expertos están operativos hasta que no sean eliminados explícitamente del gráfico; la información sobre los Asesores Expertos e indicadores de usuario adjuntos se guarda entre los inicios del terminal de cliente.

Los scripts se ejecutan sólo una vez y se eliminan automáticamente al terminar su trabajo o después de que el estado del gráfico corriente haya sido cerrado o cambiado, o después del fin de funcionamiento del terminal de cliente. Durante el reinicio del terminal de cliente los scripts no se inician porque la información sobre ellos no se guarda.

Como máximo un Asesor Experto, un script y el número ilimitado de indicadores pueden estar operativos en un gráfico.

Ejecución de programas

Cada script y cada Asesor Experto trabaja en su propio flujo de ejecución independiente. Todos los indicadores que se calculan para un símbolo trabajan en un flujo de ejecución, incluso si han sido arrancados en diferentes gráficos. De esta manera, todos los indicadores de un símbolo comparten entre ellos los recursos de un flujo de ejecución.

Todas las demás acciones relacionadas con este símbolo (procesamiento de los ticks y sincronización del historial) también se ejecutan sucesivamente en el mismo flujo junto con los indicadores. Eso quiere decir que si en un indicador se realiza una acción infinita, todos los demás eventos para este símbolo nunca se ejecutan.

Cuando se arranca un Asesor Experto, es necesario asegurar que disponga de un entorno de trading puesto al día, que pueda acceder al historial para este símbolo y período, así como realizar la sincronización entre el terminal y el servidor. Para estos procedimientos el terminal concede al EA un retraso de arranque de no más de 5 segundos, una vez expirados los cuales, el EA será iniciado con los datos que se ha podido preparar. Por eso en caso de no haber conexión con el servidor, esto puede provocar el retraso de arranque del EA.

La tabla de abajo contiene un breve sumario para los programas escritos en MQL5:

Programa	Ejecución	Nota
Script	En un flujo separado el número de flujos para los scripts coincide con el número de los scripts	Un script de ciclo cerrado no puede alterar el funcionamiento de otros programas
Asesor Experto	En un flujo separado el número de flujos para los EAs coincide con el número de los EAs	Un Asesor Experto de ciclo cerrado no puede alterar el funcionamiento de otros programas
Indicador	Un flujo de ejecución para todos los indicadores en un símbolo. El número de flujos de ejecución para los indicadores coincide con el número de los símbolos con estos indicadores	Un ciclo infinito en un indicador parará el trabajo de todos los demás indicadores en este símbolo

Inmediatamente después de que un programa haya sido adjuntado a un gráfico, este programa se carga en la memoria del terminal de cliente y se realiza la inicialización de variables globales. Si alguna variable global del tipo de clase dispone del constructor, éste va a ser invocado durante el proceso de inicialización de variables globales.

Después de todo eso el programa se encuentra en el modo de espera de un evento del terminal de cliente. Cada programa mql5 debe tener por lo menos una función-manejador de eventos, en caso contrario, el programa cargado no será ejecutado. Las funciones-manejadores de eventos tienen los nombres predefinidos, conjuntos de parámetros y los tipos de retorno predefinidos.

Tipo	Nombre de la función	Parámetros	Aplicación	Comentario
int	OnInit	no hay	expertos e indicadores	Manejador de evento Init. Se admite el tipo de valor devuelto void.
void	OnDeinit	const int reason	expertos e indicadores	Manejador de evento Deinit.
void	OnStart	no hay	scripts	Manejador de evento Start.
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[]	indicadores	Manejador de evento Calculate para todos los datos de precio.
int	OnCalculate	const int rates_total, const int prev_calculated, const int begin, const double &price[]	indicadores	Manejador de evento Calculate en un array de datos. No se permite el uso de dos manejadores Calculate en un indicador a la vez. En este caso sólo un manejador de evento Calculate va a funcionar en un array de datos.

void	OnTick	no hay	expertos	Manejador de evento NewTick. Mientras se realiza el procesamiento del evento de entrada de un nuevo tick, otros eventos de este tipo no pueden entran.
void	OnTimer	no hay	expertos e indicadores	Manejador de evento Timer.
void	OnTrade	no hay	expertos	Manejador de evento Trade.
double	OnTester	no hay		Manejador de evento Tester
void	OnChartEvent	const int id, const long &lparam, const double &dparam, const string &sparam	expertos e indicadores	Manejador de evento ChartEvent.
void	OnBookEvent	const string &symbol_name	expertos	Manejador de evento BookEvent.

El terminal de cliente envía los eventos que surgen a los correspondientes gráficos abiertos. Además, los eventos también pueden ser generados por los gráficos (eventos del gráfico) o por los programas mql5 (eventos de usuario). Usted puede activar y desactivar la generación de los eventos de creación y eliminación de los objetos gráficos ajustando las propiedades del gráfico CHART_EVENT_OBJECT_CREATE y CHART_EVENT_OBJECT_DELETE. Cada programa mql5 y cada gráfico tiene su propia cola de eventos en la que se ponen todos los eventos recién llegados.

El programa recibe los eventos sólo del gráfico en el que está iniciado. Todos los eventos se procesan uno tras otro, en orden de su llegada. Si en la cola ya hay un evento NewTick, o este evento se encuentra en el proceso de tramitación, entonces el nuevo evento NewTick no se coloca en la cola del programa mql5. Del mismo modo, si la cola del programa mql5 ya contiene un evento ChartEvent, o este evento se está procesando, el nuevo evento de este tipo no se coloca en la cola. El procesamiento del evento del temporizador se realiza según el mismo esquema. Es decir, si el evento Timer se encuentra en la cola o se está procesando, entonces el nuevo evento del temporizador no se pone en la cola.

Las colas de eventos tienen un tamaño limitado pero es suficiente, por eso la situación del desbordamiento de la cola es poco probable para un programa escrito de forma correcta. En caso del desbordamiento de la cola los nuevos eventos se descartan sin ponerse a la cola.

No se recomienda utilizar los ciclos infinitos para manejar los eventos. La excepción de esta regla podrían ser sólo los scripts que procesan sólo un único evento Start.

Las bibliotecas no manejan ningunos eventos.

Prohibición para el uso de funciones en los indicadores y EAs

Los indicadores, scripts y EAs son programas ejecutables en MQL5 y están destinados para diferentes tipos de tareas. Por esta razón existe una restricción de uso de ciertas funciones dependiendo del tipo del programa. En los indicadores están prohibidas las funciones siguientes:

- OrderCalcMargin();
- OrderCalcProfit();
- OrderCheck();
- OrderSend();
- SendFTP();
- Sleep();
- ExpertRemove();
- MessageBox().

A su vez, en los EAs y scripts están prohibidas todas las funciones destinadas para los indicadores:

- SetIndexBuffer();
- IndicatorSetDouble();
- IndicatorSetInteger();
- IndicatorSetString();
- PlotIndexSetDouble();
- PlotIndexSetInteger();
- PlotIndexSetString();
- PlotIndexGetInteger.

La biblioteca no es un programa independiente y se ejecuta en el contexto del programa MQL5 que la ha llamado: un script, indicador o EA. En consecuencia, las restricciones especificadas conciernen a la biblioteca llamada.

Carga y descarga de indicadores

Los indicadores se cargan en siguientes ocasiones:

- indicador se adjunta al gráfico;
- Inicio del terminal (si el indicador estaba adjuntado al gráfico antes del cierre anterior de terminal);
- carga de plantilla (si el indicador adjunto al gráfico está especificado en la plantilla);
- cambio de perfil (si el indicador está adjuntado a uno de los gráficos del perfil);

- cambio del símbolo o/y período del gráfico al que está adjuntado el indicador;
- después de recompilación del indicador finalizada con éxito, si este indicador ha sido adjuntado al gráfico.
- cambio de parámetros de entrada del indicador.

Los indicadores se descargan en siguientes ocasiones:

- si el indicador se desjunta del gráfico;
- cierre del terminal (si el indicador estaba adjuntado al gráfico);
- carga de plantilla, si el indicador está adjuntado al gráfico;
- cierre del gráfico al que ha sido adjuntado el indicador;
- cambio de perfil, si el indicador está adjuntado a uno de los gráficos del perfil que se cambia;
- cambio del símbolo o/y período del gráfico al que está adjuntado el indicador;
- cambio de parámetros de entrada del indicador.

Carga y descarga de Asesores Expertos

Los Asesores Expertos se cargan en siguientes ocasiones:

- Asesor Experto se adjunta al gráfico;
- Inicio del terminal (si el Asesor Experto ha sido adjuntado al gráfico antes del cierre anterior de terminal);
- carga de plantilla (si el Asesor Experto adjunto al gráfico está especificado en la plantilla);
- después de recompilación del Asesor Experto finalizada con éxito, si este Asesor Experto ha sido adjuntado al gráfico;
- cambio de perfil (si el Asesor Experto está adjuntado a uno de los gráficos del perfil);
- conexión a una cuenta, incluso si el número de la cuenta es el mismo (si el Asesor Experto ha sido adjuntado al gráfico antes de la autorización del terminal en el servidor).

La descarga del Asesor Experto adjunto al gráfico se realiza en las siguientes ocasiones:

- si el Asesor Experto se desjunta del gráfico;
- cuando el Asesor Experto se adjunta al gráfico - si ya había otro Asesor Experto en el mismo gráfico, éste se descarga ;
- el cierre del terminal (si el Asesor Experto estaba adjuntado al gráfico);
- carga de plantilla, si el Asesor Experto está adjuntado al gráfico;
- cierre del gráfico al que ha sido adjuntado el Asesor Experto;
- cambio de perfil, si el Asesor Experto está adjuntado a uno de los gráficos del perfil que se cambia;
- cambio de la cuenta a la que estaba conectado el terminal (si el Asesor Experto ha sido adjuntado al gráfico antes de la autorización del terminal en el servidor).

Si el símbolo o período del gráfico al que el Experto está adjuntado, se cambia, entonces la carga y descarga del Asesor Experto no se realiza. En este caso, sucesivamente se invocan los manejadores OnDeinit() en el símbolo/período antiguo y OnInit() en el símbolo/período nuevo (si hay), los valores de variables globales y variables estáticas no se ponen a cero. Todos los eventos que

han llegado para el Asesor Experto antes de completarse la inicialización (función OnInit()) se saltan.

Carga y descarga de scripts

Los scripts se cargan una vez adjuntados al gráfico, y se descargan inmediatamente al terminar su trabajo. Las funciones OnInit() y OnDeinit() no se invocan para los scripts.

Cuando el programa se descarga (se elimina del gráfico), ocurre deinicialización de variables globales y eliminación de la cola de mensajes. En este caso la deinicialización significa desasignación de variables del tipo string, liberación de objetos de arrays dinámicos y llamada a los destructores, en caso de que hayan.

Para mejor entendimiento de funcionamiento de los Asesores Expertos se recomienda hacer la compilación del código del Asesor Experto propuesto en el ejemplo y realizar las acciones de carga/descarga de Expertos, cambio de plantilla, símbolo, período, etc.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                  TestExpert.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://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("Initialisation");
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   Print("Deinitialisation with reason ",reason);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
```

Los scripts se cargan una vez adjuntados al gráfico, y se descargan al terminar su trabajo.

Véase también

Eventos del terminal de cliente, Funciones de procesamiento de eventos

Eventos de terminal de cliente

Init

Inmediatamente después de que el terminal de cliente cargue el programa (Asesor Experto o indicador personalizado) y arranque el proceso de inicialización de las variables globales, se enviará el evento Init que se maneja con la función OnInit() (si hay). Este evento también se genera después del cambio del instrumento financiero y/o período del gráfico, después de recompilación del programa en MetaEditor, después del cambio de parámetros de entrada desde la ventana de ajustes de Asesor Experto o indicador personalizado. Un Asesor Experto también se inicializa después del cambio de cuenta. El evento init no se genera para los scripts.

Deinit

Antes de que las variables globales se deinicialicen y el programa (Asesor Experto o indicador personalizado) se descargue, el terminal de cliente envía el evento Deinit al programa. El evento Deinit también se genera cuando el terminal de cliente finaliza su trabajo, cuando se cierra el gráfico, justo antes del cambio del instrumento financiero y/o período del gráfico, si el programa ha sido recompilado con éxito, con el cambio de parámetros de entrada, y con el cambio de cuenta.

Se puede obtener la razón de deinicialización del parámetro que ha sido pasado a la función OnDeinit (). La ejecución de la función OnDeinit() se limita con dos segundos y medio. Si la función no se ha completado en este este tiempo, su ejecución se termina de una manera forzosa. El evento Deinit no se genera para los scripts.

Start

El evento Start es un evento especial para activar un script después de cargarlo. Este evento es procesado por la función OnStart. El evento Start no se manda a los Asesores Expertos e indicadores personalizados.

NewTick

El evento NewTick se genera con la llegada de nuevas cotizaciones y se procesa por la función OnTick () de los Asesores Expertos adjuntos. Si con la llegada de nuevas cotizaciones la función OnTick está en ejecución para las cotizaciones anteriores, en este caso el Asesor Experto ignorará la cotización que ha llegado, porque el evento correspondiente no va a ponerse en la cola de eventos del Asesor Experto.

Todas las cotizaciones que llegan durante la ejecución del programa se ignoran hasta que se finalice la ejecución correspondiente de la función OnTick(). Después de eso, la función se iniciará sólo después de que se reciba una cotización nueva.

El evento OnTick() se genera independientemente de que si el comercio automático está permitido o no (el botón "Permitir/prohibir Auto trading"). La función OnTick() no se inicia con la ventana abierta de propiedades del Asesor Experto.

Calculate

El evento Calculate se genera sólo para los indicadores justo después del envío del evento init y con cualquier cambio de datos de precio. Se procesa por la función OnCalculate.

Timer

El evento Timer se genera periódicamente por el Terminal de Cliente para el Asesor Experto que ha activado el temporizador utilizando la función EventSetTimer. Habitualmente esta función se invoca en la función OnInit. El evento Timer se procesa por la función OnTimer. Una vez terminado el trabajo del Asesor Experto, hay que borrar el temporizador creado utilizando la función EventKillTimer a la que suelen llamar en la función OnDeinit.

Trade

El evento Trade se genera cuando en el servidor comercial se completa una operación comercial. La función OnTrade() maneja el evento Trade para las siguientes operaciones comerciales:

- envío, modificación o eliminación de una orden pendiente;
- cancelación de una orden pendiente por falta de medios o a la expiración del plazo de vigencia;
- activación de una orden pendiente;
- apertura, adición o cierre de una posición (o parte de posición);
- modificación de una posición abierta (cambio de stops).

TradeTransaction

Como resultado de ejecución de ciertas acciones con la cuenta de trading su estado se cambia. A estas acciones les pertenecen:

- El envío de una solicitud comercial por parte de cualquier aplicación MQL5 en el terminal de cliente utilizando la función OrderSend y OrderSendAsync, con su posterior ejecución;
- El envío de una solicitud comercial a través de la interfaz gráfica del terminal y su posterior ejecución;
- El accionamiento de órdenes pendientes y órdenes Stop en el servidor;
- La ejecución de operaciones en el servidor de trading.

Como resultado de estas acciones, para la cuenta se ejecutan las transacciones comerciales:

- tramitación de la solicitud comercial;
- cambio de órdenes abiertas;
- cambio del historial de órdenes;
- cambio del historial de operaciones;
- cambio de posiciones.

Por ejemplo, al enviar una orden de compra, ésta se tramita, para la cuenta se crea una orden de compra correspondiente, se realiza la ejecución de la orden, su eliminación de la lista de las abiertas, se agrega al historial de órdenes, luego la operación correspondiente se agrega al historial, y se crea una posición nueva. Pues todas estas acciones son transacciones comerciales. La llegada de cada una de estas transacciones al terminal es un evento TradeTransaction. Este evento se procesa con la función OnTradeTransaction.

Tester

El evento Tester se genera al terminarse el test de un Asesor Experto respecto a los datos históricos. El procesamiento del evento Tester se realiza usando la función OnTester().

TesterInit

El evento TesterInit se genera cuando se inicia el proceso de optimización en el Probador de Estrategias antes del primer repaso. El procesamiento del evento TesterInit se realiza por la función OnTesterInit().

TesterPass

El evento TesterPass se genera cuando llega un nuevo frame de datos. El procesamiento del evento TesterPass se realiza por la función OnTesterPass().

TesterDeinit

El evento TesterDeinit se genera cuando se termina el proceso de optimización del EA en el Probador de Estrategias. El procesamiento del evento TesterDeinit se realiza por la función OnTesterDeinit().

ChartEvent

Los eventos ChartEvent se generan por el Terminal de Cliente cuando el usuario trabaja con el gráfico:

- teclazo cuando la ventana del gráfico se encuentra enfocada;
- creación de objetos gráficos;
- eliminación de objetos gráficos;
- clic con ratón en un objeto gráfico que pertenece al gráfico;
- arrastre de un objeto gráfico con ratón;
- Fin de edición del texto en LabelEdit.

Además, existe un evento de usuario ChartEvent que puede ser enviado al Asesor Experto por cualquier programa mql5 utilizando la función EventChartCustom. El evento es procesado por la función OnChartEvent.

BookEvent

El terminal de cliente genera el evento BookEvent si se cambia el estado de profundidad de mercado; este evento se procesa por la función OnBookEvent. Para que el terminal de cliente empiece a generar el evento BookEvent para un símbolo especificado, es suficiente suscribirse previamente a la recepción de estos eventos para este símbolo a través de la función MarketBookAdd.

Para dar de baja la recepción del evento BookEvent para un símbolo, es necesario llamara a la función MarketBookRelease. El evento BookEvent es de difusión, lo que significa que si un Asesor Experto se suscribe a la recepción de este evento a través de la función MarketBookAdd, todos los demás Asesores que tienen el manejador OnBookEvent van a recibir este evento. Por eso hace falta analizar el nombre del símbolo que se pasa en el manejador como un parámetro.

Véase también

Funciones de procesamiento de eventos, Ejecución de programas

Recursos

El uso de la gráfica y sonidos en los programas MQL5

Los programas escritos en MQL5 permiten trabajar con los archivos de sonido e imágenes:

- PlaySound() reproduce un archivo de audio;
- ObjectCreate() permite crear las interfaces personalizadas utilizando los objetos gráficos OBJ_BITMAP y OBJ_BITMAP_LABEL.

PlaySound()

Un ejemplo de la llamada de la función PlaySound():

```
//+------------------------------------------------------------------+
//|  la función llama a la función estándar OrderSend() y reproduce  |
//|  el sonido                                                       |
//+------------------------------------------------------------------+
void OrderSendWithAudio(MqlTradeRequest  &request, MqlTradeResult &result)
  {
  //--- mandamos la solicitud al servidor
   OrderSend(request,result);
   //--- si la solicitud se acepta, reproducimos el sonido Ok.wav
   if(result.retcode==TRADE_RETCODE_PLACED) PlaySound("Ok.wav");
   //--- en caso del fallo, reproducimos el sonido de alarma desde el archivo timeout
   else PlaySound("timeout.wav");
  }
```

En este ejemplo se nos muestra cómo reproducir los sonidos desde los archivos Ok.wav y timeoit.wav que entran en la entrega estándar del terminal. Estos archivos se ubican en la carpeta **directorio_del_terminal\Sounds**. Aquí el **directorio_del_terminal** significa la carpeta desde la que ha sido iniciado el terminal de cliente MetaTrader 5. La ubicación del directorio del terminal se puede averiguar desde el programa mql5 de la siguiente manera:

```
//--- La carpeta en la que guardan los datos del terminal
   string terminal_path=TerminalInfoString(TERMINAL_PATH);
```

Usted puede utilizar los archivos de sonido no sólo desde la carpeta **directorio_del_terminal\Sounds**, sino también desde cualquier otra subcarpeta que se encuentra en la carpeta **directorio_del_terminal \MQL5.** La ubicación de la carpeta de datos del terminal en el ordenador se puede averiguar a través del menú del terminal "Archivo"-"Abrir carpeta de datos" o mediante el método programado:

```
//--- La carpeta en la que guardan los datos del terminal
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
```

Por ejemplo, si el archivo de sonido Demo.wav se encuentra en la carpeta directorio_de_datos_del_terminal\MQL5\Files, entonces la llamada de PlaySound() debe estar escrita de la siguiente manera:

```
//--- vamos a reproducir el archivo de sonido Demo.wav desde la carpeta directorio_de
```

```
   PlaySound("\\Files\\Demo.wav");
```

Preste la atención a que en el comentario la ruta del archivo está escrita con el símbolo "\", mientras que en la misma función se utiliza la secuencia "\\" para separar las carpetas dentro de la ruta.

A la hora de indicar una ruta siempre use sólo doble barra diagonal inversa como separador, puesto que una barra inversa solitaria es un símbolo de control para el compilador durante el análisis de las cadenas constantes y constantes de caracteres en el código fuente del programa.

ObjectCreate()

Ejemplo del EA que a través de la función ObjectCreate() crea el objeto "Etiqueta gráfica" (OBJ_BITMAP_LABEL).

```
string label_name="currency_label";        // nombre del objeto OBJ_BITMAP_LABEL
string euro      ="\\Images\\euro.bmp";    // ruta del archivo directorio_de_datos_de
string dollar    ="\\Images\\dollar.bmp";  // ruta del archivo directorio_de_datos_de
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- creamos el botón OBJ_BITMAP_LABEL si no lo tenemos todavía
   if(ObjectFind(0,label_name)<0)
     {
      //--- intentaremos crear el objeto OBJ_BITMAP_LABEL
      bool created=ObjectCreate(0,label_name,OBJ_BITMAP_LABEL,0,0,0);
      if(created)
        {
         //--- enlazamos el botón a la esquina superior derecha del gráfico
         ObjectSetInteger(0,label_name,OBJPROP_CORNER,CORNER_RIGHT_UPPER);
         //--- ahora ajustaremos las propiedades del objeto
         ObjectSetInteger(0,label_name,OBJPROP_XDISTANCE,100);
         ObjectSetInteger(0,label_name,OBJPROP_YDISTANCE,50);
         //--- pondremos a 0 el código del último error
         ResetLastError();
         //--- cargaremos la imagen para el estado del botón "Pulsado"
         bool set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,0,euro);
         //--- comprobaremos el resultado
         if(!set)
           {
            PrintFormat("Fallo al cargar la imagen desde el archivo %s. Código del er
           }
         ResetLastError();
         //--- cargaremos la imagen para el estado del botón "Despulsado"
         set=ObjectSetString(0,label_name,OBJPROP_BMPFILE,1,dollar);

         if(!set)
```

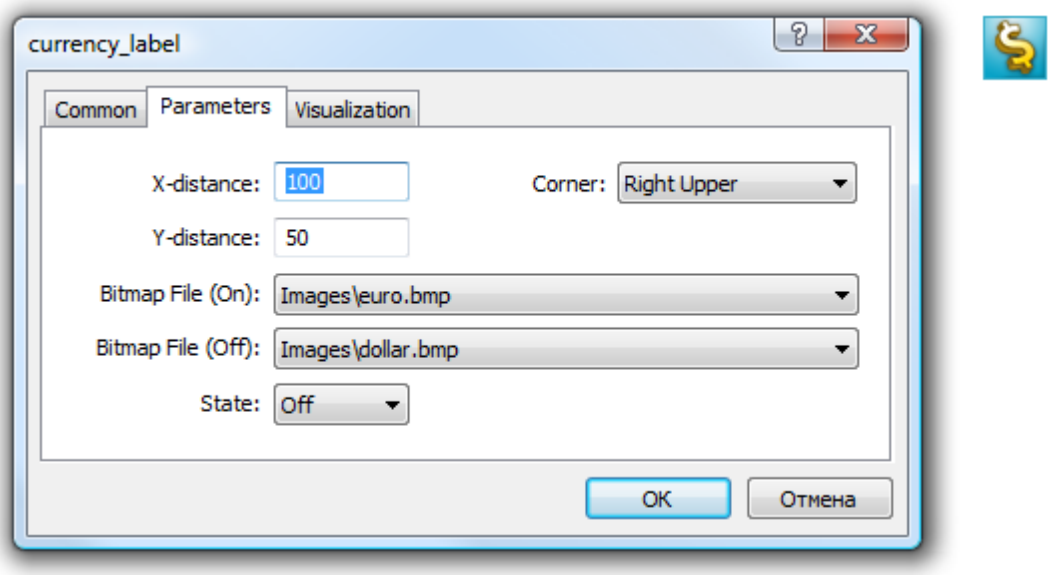
```
            {
             PrintFormat("Fallo al cargar la imagen desde el archivo %s. Código del er
            }
          //--- enviaremos al gráfico el comando de reinicio para que el botón aparezc
          ChartRedraw(0);
         }
       else
         {
          //--- fallo al crear el objeto, avisaremos sobre ello
          PrintFormat("Fallo al crear el objeto OBJ_BITMAP_LABEL. Código del error %d"
         }
      }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- quitaremos el objeto desde el gráfico
   ObjectDelete(0,label_name);
  }
```

La creación y configuración del objeto gráfico con el nombre currency_label se llevan a cabo en la función OnInit(). Las rutas hacia los archivos de imágenes se establecen en las variables globales **euro** y **dollar**, como separador se utiliza la barra diagonal inversa doble:

```
string euro      ="\\Images\\euro.bmp";    // ruta del archivo directorio_de_datos_de
string dollar    ="\\Images\\dollar.bmp";  // ruta del archivo directorio_de_datos_de
```

En este caso los archivos se encuentran en la carpeta **directorio_de_datos_del_terminal**\MQL5\Images.

En realidad el objeto OBJ_BITMAP_LABEL representa un botón que en función de su estado (pulsado o despulsado) puede visualizar una de las dos imágenes: euro.bmp o dollar.bmp.



El tamaño del botón con la interfaz gráfica se ajusta automáticamente al tamaño de la imagen a mostrar. La imagen se cambia con el clic izquierdo en el objeto OBJ_BITMAP_LABEL (**en las propiedades tiene que estar seleccionada la opción "Desactivar la selección"**). El objeto OBJ_BITMAP se crea de la misma manera y sirve para crear el fondo con la imagen necesaria.

El valor de la propiedad OBJPROP_BMPFILE que responde de la apariencia de los objetos OBJ_BITMAP y OBJ_BITMAP_LABEL se puede cambiar de forma dinámica. Esto permite crear diferentes interfaces personalizadas interactivas para los programas mql5.

Inserción de recursos en los archivos ejecutables durante la compilación de programas mql5

Un programa mql5 a lo mejor podrá necesitar varios diferentes recursos cargables en forma de archivos de imágenes o sonidos. Para evitar la necesidad de transferir todos estos archivos durante el movimiento del programa ejecutable en MQL5, se debe utilizar la directiva #resource:

```
#resource ruta_hacia_archivo_del_recurso
```

El comando #resource indica al compilador que hay que hay que incluir el recurso según la ruta especificada **ruta_hacia_archivo_del_recurso** en el archivo ejecutable EX5. De esta manera, se puede colocar todas las imágenes y sonidos necesarios directamente en el archivo EX5, sin tener que pasar todos los archivos que utiliza el programa para que funcione en otro terminal. Cualquier archivo EX5 puede contener recursos, y cualquier programa EX5 puede utilizar los recursos desde otro programa EX5.

Los archivos en el formato BMP y WAV se comprimen automáticamente antes de ser insertados en el archivo ejecutable EX5. Esto quiere decir que el uso de los recursos no sólo permite crear los programas MQL5 de pleno valor sino también reduce el tamaño total de los archivos requeridos por el terminal a la hora de utilizar la gráfica y el audio en comparación con el modo común de creación de los programas mql5.

El tamaño del archivo de un recurso no puede superar 16 Mb.

Búsqueda de recursos especificados por el compilador

Un recurso puede ser insertado mediante el comando #resource "<ruta_hacia_archivo_del_recurso>"

```
#resource "<ruta_hacia_archivo_del_recurso>"
```

La longitud de la cadena constante <ruta_hacia_archivo_del_recurso> no puede ser más de 63 caracteres.

El compilador busca el recurso especificado en orden siguiente:

- si la ruta se empieza con la barra inversa separadora "\" (se escribe "\\"), entonces el recurso se busca respecto al catálogo **carpeta_de_datos_del_terminal**\MQL5\,
- si no hay ninguna barra inversa, el recurso se busca respecto a la ubicación del archivo fuente en el que este recurso ha sido insertado.

En la ruta del recurso no se puede utilizar las subcadenas "..\\" y ":\\".

Algunos ejemplos de inserción de recursos:

```
//--- la especificación correcta de los recursos
#resource "\\Images\\euro.bmp" // euro.bmp se encuentra en carpeta_de_datos_del_termi
#resource "picture.bmp"        // picture.bmp se encuentra en la misma carpeta que el
#resource "Resource\\map.bmp"  // recurso se encuentra en la carpeta catálogo_del_arc

//--- indicación incorrecta de los recursos
#resource ":picture_2.bmp"     // no se puede utilizar ":"
#resource "..\\picture_3.bmp"  // no se puede utilizar ".."
#resource "\\Files\\Images\\Folder_First\\My_panel\\Labels\\too_long_path.bmp" //más
```

Uso de recursos

Nombre del recurso

Después de que el recurso haya sido declarado mediante la directiva #resource, puede utilizarlo en cualquier parte del programa. El nombre del recurso será su ruta sin la barra inversa al principio de la línea que establece la ruta del archivo. Para poder utilizar su propio recurso en el código, hay que añadir el signo especial "::" antes del nombre de este recurso.

Ejemplos:

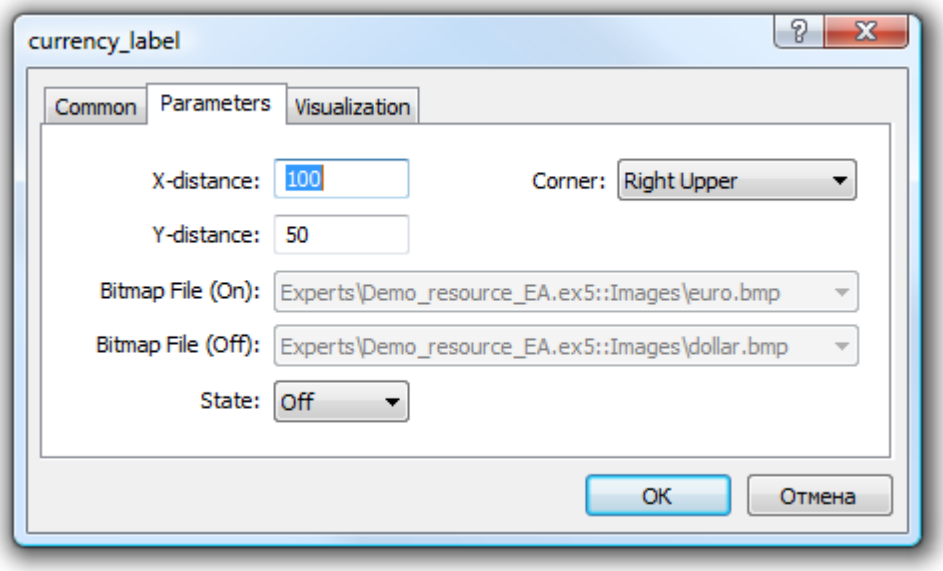
```
//--- ejemplos de indicación de recursos y sus nombres en el comentario
#resource "\\Images\\euro.bmp"          // nombre del recurso - Images\euro.bmp
#resource "picture.bmp"                 // nombre del recurso - picture.bmp
#resource "Resource\\map.bmp"           // nombre del recurso - Resource\map.bmp
#resource "\\Files\\Pictures\\good.bmp" // nombre del recurso - Files\Pictures\good.b
#resource "\\Files\\Demo.wav";          // nombre del recurso - Files\Demo.wav"
#resource "\\Sounds\\thrill.wav";       // nombre del recurso - Sounds\thrill.wav"
...
```

```
//--- uso de recursos
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\\good.bmp");
...
PlaySound("::Files\\Demo.wav");
...
PlaySound("::Sounds\\thrill.wav");
```

Cabe mencionar que cuando se establece una imagen desde el recurso para los objetos OBJ_BITMAP y OBJ_BITMAP_LABEL, el valor de la propiedad OBJPROP_BMPFILE ya no se puede cambiar manualmente. Por ejemplo, estamos utilizando los recursos de los archivos euro.bmp y dollar.bmp para crear OBJ_BITMAP_LABEL.

```
#resource "\\Images\\euro.bmp";    // euro.bmp se encuentra en carpeta_de_datos_del_t
#resource "\\Images\\dollar.bmp";  // dollar.bmp se encuentra en carpeta_de_datos_del
```

Entonces si nos fijamos en las propiedades de este objeto, veremos que las propiedades BitMap File (On) y BitMap File (Off) tienen el color gris y no están disponibles para el cambio manual:



Uso de recursos de otros programas mql5

El uso de los recursos también tiene otra ventaja - en cualquier programa mql5 se puede utilizar los recursos desde cualquier archivo EX5. De esta manera, los recursos desde un archivo EX5 se puede utilizar en muchos otros programas mql5.

Para poder usar el nombre del recurso desde otro archivo, hay que indicarlo como sigue <ruta_nombre_del_archivo_EX5>::<nombre_del_recurso>. Por ejemplo, supongamos que el script Draw_Triangles_Script.mq5 contiene el recurso para una imagen en el archivo triangle.bmp:

```
#resource "\\Files\\triangle.bmp"
```

Entonces su nombre para el uso en el mismo script será el siguiente "Files\triangle.bmp", y para poder usarlo hay que añadir a su nombre el signo especial "::".

```
//--- el uso del recurso en el mismo script
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"::Files\\triangle.bmp");
```

Para tener la posibilidad de usar el mismo recurso desde otro programa, por ejemplo desde un Asesor Experto, hay que añadir al nombre del recurso la ruta del archivo EX5 respecto a la carpeta **carpeta_de_datos_del_terminal\MQL5** y el nombre del archivo EX5 de este script - **Draw_Triangles_Script.ex5**. Supongamos que el script se encuentra en la carpeta estándar **carpeta_de_datos_del_terminal\MQL5**\Scripts\, entonces la llamada hay que escribir de la siguiente manera:

```
//--- el uso del recurso del script en el Asesor Experto
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"\\Scripts\\Draw_Triangles_Script.
```

Si durante la llamada al recurso de otro archivo EX5 no indicamos la ruta de este archivo ejecutable, entonces la búsqueda de este archivo ejecutable se realiza en la misma carpeta donde se encuentra el programa que ha llamado al recurso. Eso quiere decir que si en el Asesor Experto se llama al recurso desde el archivo Draw_Triangles_Script.ex5 sin especificar la ruta, por ejemplo así:

```
//--- llamada al recurso del script en el EA sin especificar la ruta
ObjectSetString(0,my_bitmap_name,OBJPROP_BMPFILE,0,"Draw_Triangles_Script.ex5::Files\
```

entonces el archivo va a buscarse en la carpeta **carpeta_de_datos_del_terminal\MQL5**\Experts\ si el mismo EA se encuentra en la carpeta **carpeta_de_datos_del_terminal\MQL5**\Experts\.

Véase también

ResourceCreate(), ResourceSave(), PlaySound(), ObjectSetInteger(), ChartApplyTemplate(), Operaciones con archivos

Llamadas a las funciones importadas

Para importar funciones durante la ejecución de un programa mql5 se utiliza la ligadura temprana. Eso significa que si en el programa hay llamada a una función importada, el módulo correspondiente (ex5 o dll) se carga durante la carga del programa. Las bibliotecas MQL5 y DLL se ejecutan en el flujo del módulo-invocador.

No se recomienda utilizar el nombre completamente especificado del módulo a cargar como el *Drive:\Directory\FileName.Ext*. Las bibliotecas MQL5 se cargan de la carpeta *terminal_dir\MQL5\Libraries*. Si no encontramos ninguna, intentamos cargar la biblioteca desde la carpeta *terminal_dir\experts*.

Las bibliotecas de sistema (DLL) se cargan según las reglas del sistema operativo. Si la biblioteca ya está cargada (por ejemplo, por otro Asesor Experto, e incluso de otro terminal de cliente inicializado en paralelo), entonces nos dirigimos a la biblioteca ya cargada. En caso contrario, la búsqueda se efectua como sigue:

1. Directorio del que ha sido iniciado el módulo que importa dll. Hablando del módulo, se quiere decir un Asesor Experto, un script, un indicador o biblioteca EX5;
2. Directorio directorio_de_terminal_datos\MQL5\Libraries (TERMINAL_DATA_PATH\MQL5\Libraries);
3. Directorio del que ha sido iniciado el terminal de cliente de MetaTrader 5;
4. Directorio de sistema;
5. Directorio Windows;
6. Directorio corriente;
7. Directorios enumerados en la variable de sistema PATH.

Si una biblioteca DLL utiliza en su trabajo otra biblioteca, la primera no puede ser cargada si falta la segunda DLL.

Antes de que se cargue un Asesor Experto (script, indicador) se forma una lista general de todos los módulos bibliotecarios EX5. Se prevé usarlos como del Asesor Experto cargado (script, indicador), tanto de las bibliotecas de esta lista. De esta manera cargándose sólo una vez, los módulos bibliotecarios EX5 se utilizan varias veces. Las bibliotecas usan las variables predefinidas del Asesor Experto (script, indicador) que las ha invocado.

El orden de búsqueda de la biblioteca EX5 importada es el siguiente:

1. Directorio, ruta al cual se establece con relación al directorio del Asesor Experto (script, indicador) que importa EX5;
2. Directorio directorio_de_terminal\MQL5\Libraries;
3. Directorio MQL5\Libraries en el directorio general de todos los terminales de cliente de MetaTrader 5(Common\MQL5\Libraries).

Las funciones importadas de DLL en el programa mql5 deben asegurar el acuerdo de enlaces para las funciones Windows API. Para asegurar dicho acuerdo en el texto fuente de programas escritos en C o C++ se utiliza la palabra clave __stdcall, bastante específica para los compiladores de Microsoft(r). El acuerdo en cuestión se caracteriza por lo siguiente:

- función que invoca (en nuestro caso es el programa mql5) tiene que "ver" el prototipo de función invocada (importada de DLL) para colocar los parámetros en la pila de una forma correcta;
- función que invoca (en nuestro caso es un programa mql5) coloca los parámetros en la pila de una

forma inversa - de derecha a izquierda; precisamente en este orden una función importada lee los parámetros traspasados para ella;

- parámetros se traspasan por valor, salvo los que se pasan explícitamente por referencia (en nuestro caso, líneas);
- función importada limpia la pila cuando lee losparámetros pasados para ella.

A la hora de describir el prototipo de una función importada se puede utilizar los parámetros con valores por defecto.

Si la biblioteca correspondiente no ha podido cargarse o si está prohibido usar DLL, o no se ha encontrado la función importada, entonces el Asesor Experto deja de funcionar dejando el mensaje "expert stopped" en el registrador de datos. El Asesor Experto se cargará hasta que no vuelva a ser inicializado. El Asesor Experto podrá ser reinicializado después de recompilación o después de que abramos la tabla de propiedades de Asesor y apretemos el botón "Ok".

Traspaso de parámetros

Todos los parámetros de tipos simples se traspasan por valor, si no se indica explícitamente que se pasan por referencia. Cuando se traspasa la cadena, se traspasa la dirección del buffer de la cadena copiada; si la cadena se pasa por referencia , en la función importada de DLL se pasa la dirección del buffer precisamente de esta cadena sin copiar.

Las estructuras que contienen los arrays dinámicos, cadenas, clases, otras estructuras complejas, así como los arrays dinámicos o estáticos de los objetos mencionados, no pueden ser traspasadas en la función importada en calidad de parámetros.

Cuando se traspasa un array en DLL, siempre (haya o no haya la bandera AS_SERIES) se pasa la dirección del inicio del búfer de datos. La función dentro de DLL no sabe nada de la bandera AS_SERIES, el array traspasado es un array estático de una longitud desconocida, y para especificar el tamaño del array se utiliza un parámetro adicional.

Errores de ejecución

En el subsistema ejecutivo del terminal de cliente existe posibilidad de guardar el código de error en caso de que éste surja durante la ejecución del programa mql5. Hay una variable predefinida _LastError para cada programa mql5 ejecutable.

La variable _LastError se pone a cero antes del inicio de la función OnInit. En caso de surgir una situación errónea durante los cálculos o a la hora de llamar a una función built-in, la variable _LastError acepta el código correspondiente del error. El valor guardado en esta variable se puede obtener utilizando la función GetLastError().

Existe una serie de errores críticos, y cuando éstos surgen la ejecución del programa se detiene inmediatamente:

- división por cero;
- salida fuera de los límites del array;
- uso incorrecto de un puntero a objeto;

Simulación de estrategias comerciales

La idea del trading automatizado es bastante atractiva con el hecho de que el robot de trading trabaja sin descanso 24 horas al día y siete días a la semana. El robot no sabe nada de cansancio, dudas y miedo, ni tampoco de problemas psicológicos. Sólo basta con formalizar las reglas de trading e implementarlas en forma de algoritmos, y su robot está listo a trabajar sin parar. Pero antes, es necesario asegurarse del cumplimiento de dos condiciones importantes:

- el Asesor Experto realiza las operaciones comerciales de acuerdo con las reglas del sistema de trading;
- la estrategia de trading que ha sido implementada en el Asesor Experto muestra la ganancia a base de los datos históricos.

Para recibir respuestas a estas preguntas, se utiliza el Probador de Estrategias que forma parte integrante del Terminal de Cliente MetaTrader 5.

En este apartado vamos a analizar todas las particularidades de la simulación y optimización de los programas en el Probador de Estrategias:

- Modos de generación de ticks
- Modelación de spreads
- Variables globales del Terminal de Cliente
- Cálculo de indicadores durante la simulación
- Carga del historial durante la simulación
- Simulación en múltiples divisas
- Modelación de la hora en el Probador
- Objetos gráficos durante la simulación
- Función OnTimer() en el Probador
- Función Sleep() en el Probador
- Función Print() en el Probador
- Uso del Probador para las tareas de optimización en los cálculos matemáticos
- Sincronización de las barras durante la simulación en el modo "Sólo precios de apertura"
- Función IndicatorRelease() en el Probador
- Procesamiento de eventos en el Probador
- Agentes de pruebas
- Intercambio de datos entre el Terminal y el Agente
- Uso de la carpeta compartida de todos los Terminales de Cliente
- Uso de la DLL

Modos de generación de ticks

Un Asesor Experto escrito en el lenguaje MQL5 es un programa que se inicia cada vez como respuesta a una acción externa - un evento. Para cada evento predefinido el EA dispone de una función correspondiente a este evento - manejador de eventos.

El evento más importante para un EA es el cambio del precio - NewTick. Por esta razón, para la

simulación de los EAs es necesario generar las secuencias de ticks. En el Probador del Terminal de Cliente MetaTrader 5 hay 3 modos de generación de ticks:

- Todos los ticks
- Precios OHLC de las barras de un minuto (1 Minute OHLC)
- Sólo precios de apertura

El modo "Todos los ticks" es el modo básico y el más detallado de generación de los ticks, los dos restantes es una simplificación del modo principal y serán descritos en comparación con el modo "Todos los ticks". Vamos a analizar los tres modos para comprender la diferencia entre ellos.

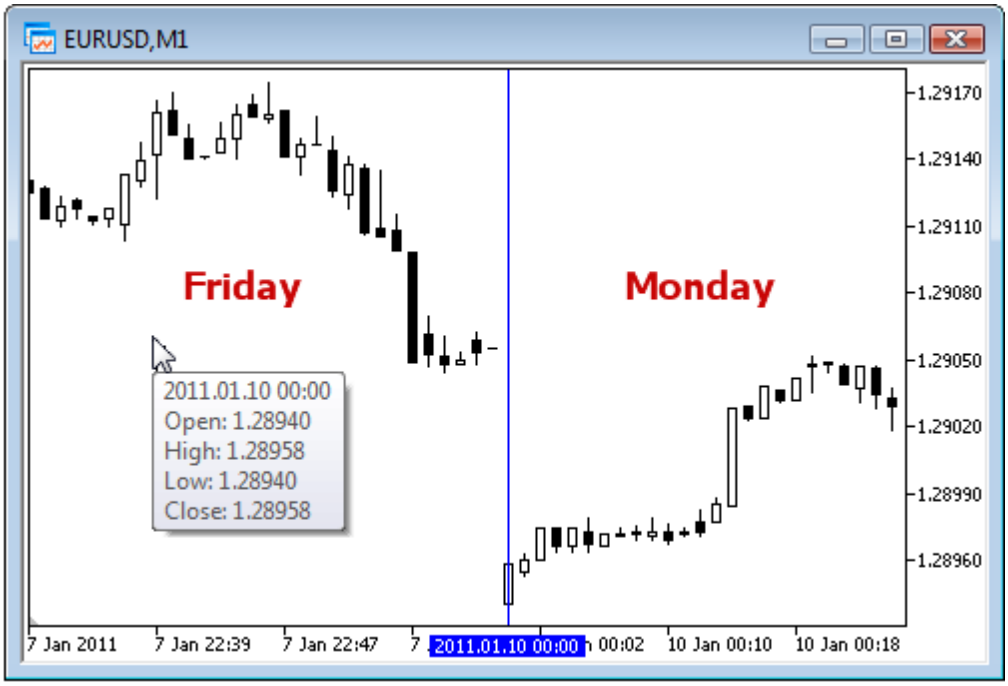
Todos los ticks

El historial de cotizaciones de los instrumentos financieros se traspasa del servidor comercial al Terminal de Cliente MetaTrader 5 en forma de los bloques de barras de un minuto bien comprimidos. Usted puede encontrar la información detallada sobre cómo se hace la solicitud y construcción de los períodos necesarios en el apartado de la ayuda Organización de acceso a los datos.

El elemento mínimo del historial de precios es una barra de un minuto desde la que se puede conseguir la información sobre los valores de cuatro precios:

- Open - precio de apertura de la barra de un minuto;
- High - el máximo alcanzado durante esta barra de un minuto;
- Low - el mínimo alcanzado durante esta barra de un minuto;
- Close - precio de cierre de la barra.

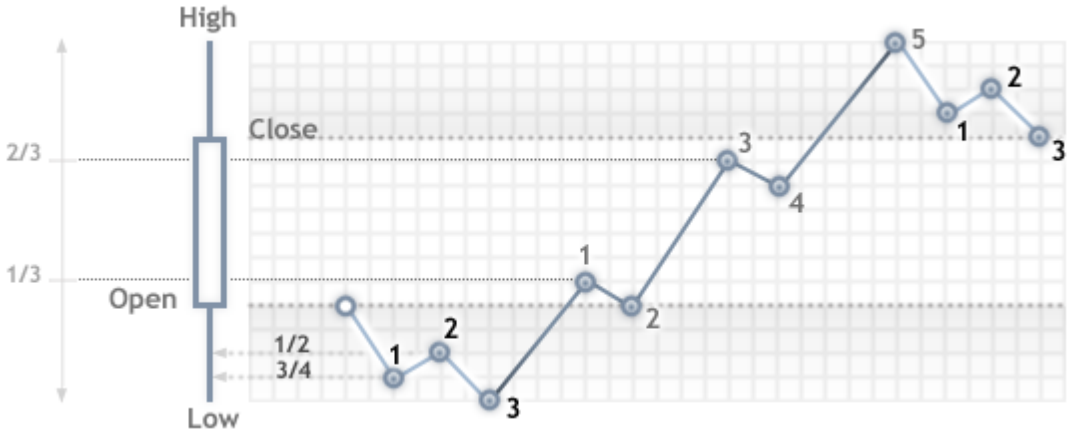
La nueva barra de un minuto no se abre en el momento cuando se empieza un nuevo minuto (el número de segundos llega a ser 0), sino cuando llega un tick nuevo, es decir, cuando el precio se cambia por lo menos en un punto. En la imagen se muestra la primera barra minutera de nueva semana de trading con fecha y hora de apertura 2011.01.10. 00:00. La diferencia de precios entre el viernes y el lunes que vemos en el gráfico es un hecho corriente, puesto que incluso durante los días de descanso las cotizaciones de divisas van cambiando en respuesta a las noticias que llegan.



Respecto a esta barra minutera sólo sabemos que fue abierta el 10 de Enero de 2011 a las 00:00, pero no sabemos nada de los segundos. Esto podía pasar a las 00:00:12 o 00:00:36 (pasados 12 o 36 segundos desde el inicio de la nueva jornada), o cualquier otro momento dentro de este minuto. Pero lo que sabemos exactamente es que en el momento de apertura de la nueva barra minutera el precio Open para EURUSD se encontraba en 1.28940.

Igualmente, tampoco sabemos con una precisión de un segundo cuándo ha llegado el tick correspondiente al precio de cierre de la barra en cuestión. Lo único que sabemos es que se trata del último precio en esta barra minutera que ha sido apuntado como el precio Close. Para este minuto este precio es 1.28958. El tiempo de aparición de los precios High y Low tampoco se sabe. Pero sabemos que el precio máximo y el mínimo ha alcanzado sin duda alguna los niveles 1.28958 y 1.28940, respectivamente.

Para probar la estrategia comercial nos hace falta una secuencia de ticks sobre la que va a emularse el trabajo del EA. De esta manera, para cada barra de un minuto sabemos **4 puntos de control** sobre los que podemos decir con total seguridad que el precio ha estado ahí. Si una barra tiene sólo 4 ticks, esta información será suficiente para la simulación, pero normalmente el volumen de tick es superior a 4. Eso significa que hace falta generar los puntos de control adicionales para los ticks que han llegado entre los precios Open, High, Low y Close. El principio de generación de los ticks en el modo "Todos los ticks" se describe en el artículo Algoritmo de generación de los ticks en el Probador de Estrategias del terminal MetaTrader 5 la ilustración desde el cual se muestra más abajo.



Durante la simulación el modo "Todos los ticks" la función OnTick() del EA va a **llamarse en cada punto de control**, siendo cada punto de control un tick desde la secuencia generada. El EA va a recibir la hora y el precio del tick modelado igualmente como durante el trabajo en tiempo real.

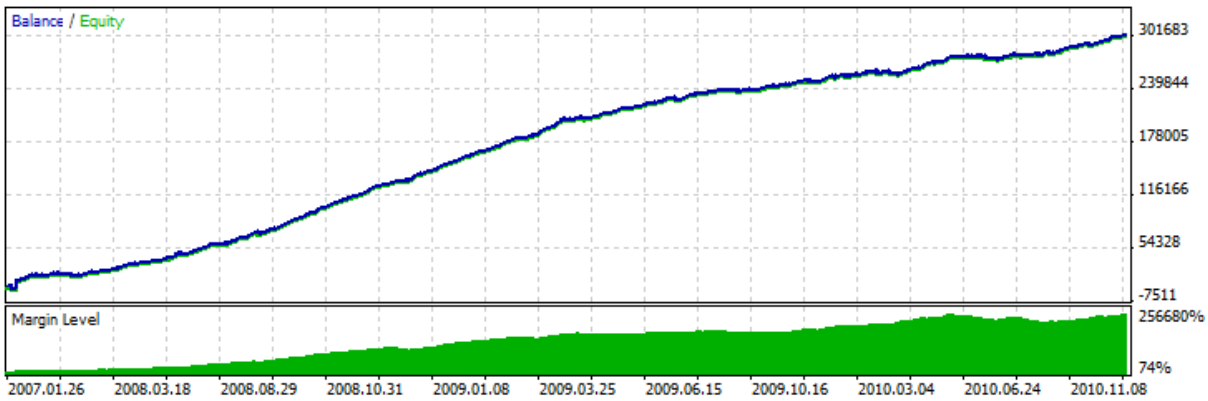
> **Importante:** el modo de simulación "Todos los ticks" es el más preciso pero también es el más duradero. Para una valoración primaria de la mayoría de las estrategias comerciales generalmente es suficiente utilizar alguno de otros dos modos de prueba.

1 minute OHLC

El modo de simulación "Todos los ticks" es el más preciso de los tres, pero al mismo tiempo es el más lento. La función OnTick() se inicia para cada tick, y el volumen de tick puede ser bastante grande. Para las estrategias a las que no les importa en qué secuencia de ticks se desarrollaba el precio en el transcurso de cada barra existe el modo de modelación más rápido y más aproximado - "1 minute OHLC".

En el modo "1 minute OHLC" la secuencia de ticks se construye sólo a base de los **precios OHLC de las barras de un minuto**, en este caso el número de puntos de control generados se reduce sustancialmente. Por consiguiente, se reduce la duración de la prueba. El inicio de la función OnTick() se hace en todos los puntos de control que se construyen a base de los precios OHLC de las barras minuteras.

La renuncia a generar los ticks intermedios adicionales entre los precios Open, High, Low y Close hace que aparece una determinación fuerte en el desarrollo del precio a partir del momento en el que ha sido determinado el precio Open. Esto da posibilidades para crear el "Grial de simulación" que muestra durante la simulación un bonito gráfico ascendiente. Puede encontrar un ejemplo de este Grial en Code Base - Grr-al.



En la imagen podemos ver un gráfico muy atractivo de simulación de este EA. ¿Cómo ha salido así? Para una barra minutera se saben 4 precios. Y es sabido con seguridad que el precio Open va primero, y el último es el precio Close. Entre ellos hay precios High y Low, el orden de su aparición no se sabe, pero se sabe que el precio High es mayor o igual al precio Open (el precio Low es menor o igual al precio Open).

Basta con averiguar el momento de llegada del precio Open y luego analizar el siguiente tick con el fin de determinar qué es lo que tenemos delante, ¿High o Low? Si el precio es más bajo que el precio Open, eso significa que tenemos delante de nosotros el precio Low - entonces hacemos la compra en este tick. El siguiente tick va a corresponder al precio High en el que cerramos la compra y abrimos la venta. El siguiente tick es el último - es el precio Close, cerramos la venta en este tick.

Si después del precio ha llegado un tick con el precio que es más alto que el precio de apertura, entonces la secuencia de transacciones es inversa. Vamos a procesar en este modo tramposo la barra de un minuto y esperamos la siguiente. Mientras probamos este EA en los datos históricos, todo va perfectamente. Pero cuando lo hagamos en tiempo real, el cuento de hadas se desvanece -la línea del balance sigue siendo recta pero va hacia abajo. Para desvelar el truco, sólo hay que probar este EA en el modo "Todos los ticks".

> **Importante:** si los resultados de simulación del EA con el uso de los modos aproximados ("1 minute OHLC" y "Sólo precios de apertura") salen muy buenos, haga la prueba obligatoriamente en el modo "Todos los ticks".

Sólo precios de apertura

En este modo se generan los ticks sobre los precios OHLC del período (timeframe) seleccionado para la simulación. En este caso la función OnTick() del EA se inicia sólo al principio de la barra para el precio Open. Gracias a esta particularidad los niveles stop y las órdenes pendientes pueden iniciarse por el precio diferente al especificado (sobre todo durante la simulación en los períodos mayores). A cambio de eso, tenemos la posibilidad de llevar a cabo la simulación estimativa del EA de forma bastante rápida.

La excepción durante la simulación de ticks en el modo "Sólo precios de apertura" son los períodos W1 y MN1: para estos períodos los ticks se generan para los precios OHLC de cada día, y no para los precios OHLC de la semana y mes, respectivamente.

Por ejemplo, se hace la prueba del EA para EURUSD H1 en el modo "Sólo precios de apertura". En este caso, el número total de los ticks (puntos de control) no va a superar 4*número de barras de una hora que están dentro del intervalo de simulación. Pero la **llamada al manejador OnTick() se hace sólo en**

el momento de la apertura de la **barra de una hora**. En los demás ticks ("invisibles" para el EA) se hacen las comprobaciones necesarias para la simulación correcta:

- cálculo de requerimientos de margen;
- accionamiento de Stop Loss y Take Profit;
- accionamiento de órdenes pendientes;
- eliminación de órdenes pendientes caducadas.

Si no hay posiciones abiertas u órdenes pendientes, entonces tampoco hay necesidad en estas comprobaciones para los ticks invisibles, y el aumento de la velocidad puede llegar a ser bastante importante. El modo "Sólo precios de apertura" conviene muy bien para probar las estrategias que realizan las transacciones sólo en la apertura de la barra, y no utilizan las órdenes pendientes ni tampoco las órdenes StopLoss,. TakeProfit. Para el tipo de estas estrategias se mantiene toda la precisión de simulación necesaria.

Como ejemplo de un EA para el que no importa el modo de simulación vamos a mostrar el EA Moving Average de la entrega estándar. La lógica de este EA está construida de tal manera que todas las decisiones se toman en la apertura de la barra y las transacciones se realizan enseguida, sin utilizar las órdenes pendientes. Vamos a arrancar la prueba del EA para EURUSD H1 en el intervalo desde 2010.01.09 hasta 2010.31.12, y compararemos los gráficos. En la imagen se muestran los gráficos del balance desde el informe del Probador para los tres modos.

Como puede ver, los gráficos de diferentes modos de simulación son absolutamente idénticos para el EA Moving Average desde la entrega estándar.

Existen ciertas limitaciones de la aplicación del modo "Sólo precios de apertura":

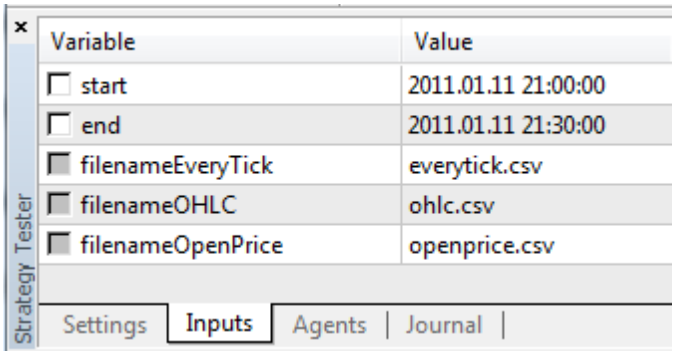
- No se puede utilizar el modo de trading "Retraso aleatorio";
- En el EA que se prueba no es posible acceder a los datos del inferior período que se utiliza para la simulación/optimización. Por ejemplo, si para la simulación/optimización se utiliza el período H1, Usted puede acceder a los datos del H2, H3, H4, etc., y no a los del M30, M20, M10, etc. Aparte de eso, los períodos mayores a los que se accede tienen que ser múltiplos del período de simulación.

Por ejemplo, durante la simulación en el período M20 no se puede acceder al período M30, pero se puede dirigirse al H1. Estas limitaciones están condicionadas a la imposibilidad de obtener los datos de los períodos inferiores y no múltiplos desde las barras que se generan durante la simulación/optimización.

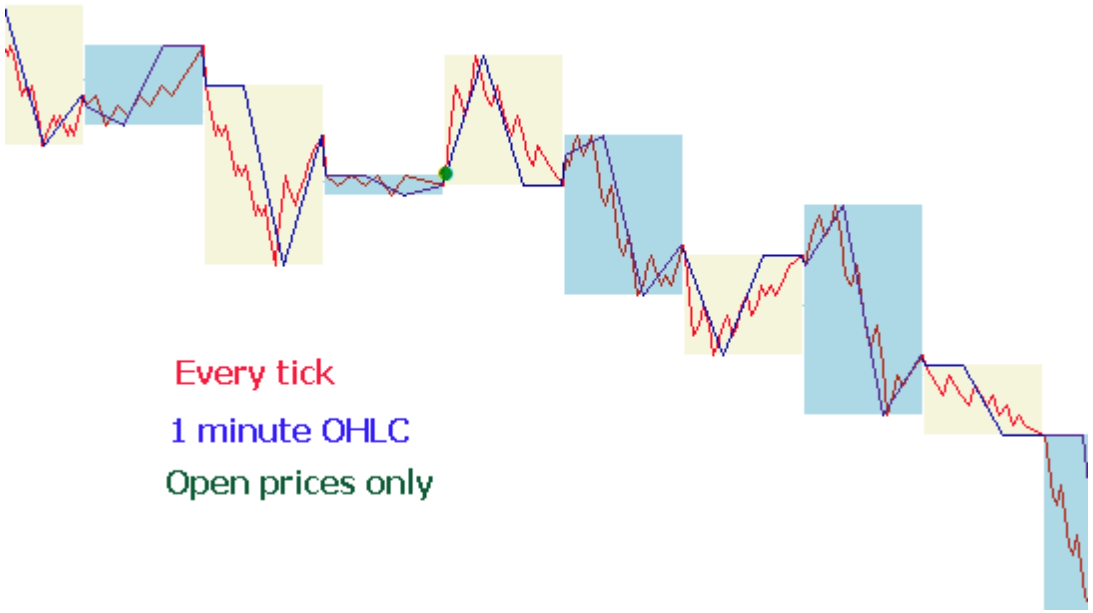
- Las limitaciones del acceso a los datos de otros períodos se extienden también a otros símbolos cuyos datos utiliza el EA. No obstante, en este caso la limitación para cada símbolo depende del primer período al que se ha accedido durante la simulación/optimización. Por ejemplo, la simulación se lleva a cabo para el símbolo y período EURUSD H1, el EA se ha dirigido por primera vez al símbolo GBPUSD M20. En esta situación el EA puede utilizar en adelante los datos del EURUSD H1, H2, etc., así como los del GBPUSD M20, H1, H2, etc.

Importante: el modo "Sólo precios de apertura" es el más rápido por el tiempo que dura la simulación, pero no conviene para todas las estrategias de trading. Se debe seleccionar el modo de simulación necesario en función de las particularidades del sistema de trading que se utiliza.

Al final de esta sección que trata sobre los modos de modelación vamos a mostrar la comparación visual de diferentes modos de generación de ticks para EURUSD para dos barras de M15 en el intervalo 2011.01.11 21:00:00 - 2011.01.11 21:30:00. Los ticks ha sido escritos en archivos diferentes, utilizando el EA WriteTicksFromTester.mq5. La terminación de los nombres de estos archivos se establecen en los parámetros input filenamEveryTick, filenameOHLC y filenameOpenPrice.



Para obtener tres archivos con tres secuencias de ticks (para cada uno de los modos "Todos los ticks", "OHLC en barras minuteras" y "Sólo precios de apertura") el EA ha sido arrancado tres veces en los modos correspondientes en repasos únicos. Luego, utilizando el indicador TicksFromTester.mq5, los datos han sido proyectados al gráfico desde estos tres archivos. El código del indicador va adjunto al artículo.



Por defecto, todas las operaciones con archivos en el lenguaje MQL5 se realizan dentro de los límites de una "sandbox de archivos", y durante la simulación el EA tiene disponible sólo su propia "sandbox de archivos". Para que el indicador y el EA puedan trabajar durante la simulación con los archivos de la misma carpeta, se utiliza la bandera FILE_COMMON. Ejemplo del código del EA:

```
//--- abrimos el archivo
   file=FileOpen(filename,FILE_WRITE|FILE_CSV|FILE_COMMON,";");
//--- comprobamos el éxito de la operación
   if(file==INVALID_HANDLE)
     {
      PrintFormat("No se ha podido abrir el archivo %s para la escritura. Código del
      return;
     }
   else
     {
      //--- avisamos sobre el guardado en la carpeta compartida de todos los terminal
      PrintFormat("El archivo será guardado en la carpeta %s",TerminalInfoString(TERM
     }
```

En el indicador para la lectura de datos también se ha utilizado la bandera FILE_COMMON, lo que ha permitido evitar el traspaso manual de los archivos necesarios de una carpeta a otra.

```
//--- abrimos el archivo
   int file=FileOpen(fname,FILE_READ|FILE_CSV|FILE_COMMON,";");
//--- comprobamos el éxito de la operación
   if(file==INVALID_HANDLE)
     {
      PrintFormat("No se ha podido abrir el archivo %s para la lectura. Código del er
      return;
     }
   else
     {
      //--- indicaremos la ubicación de la carpeta compartida de todos los terminales
      PrintFormat("El archivo será leído desde la carpeta %s",TerminalInfoString(TERM
     }
```

Modelación de spreads

La diferencia entre los precios Bid y Ask se llama spread. El spread no se modela durante la simulación, sino se coge de los datos históricos. Si en los datos históricos el spread es más bajo o igual a cero, entonces se utiliza el spread actual al momento de solicitar la información por parte del agente de pruebas.

En el Probador un spread siempre se considera como flotante. Es decir, SymbolInfoInteger(symbol, SYMBOL_SPREAD_FLOAT) siempre devuelve true.

Además, en los datos históricos se guardan los valores de los volúmenes de ticks y de los volúmenes comerciales. Para almacenar y obtener los datos, se utiliza una estructura especial MqlRates:

```
struct MqlRates
  {
   datetime time;         // fecha/hora de apertura de la barra
   double   open;         // precio de apertura Open
   double   high;         // precio máximo High
   double   low;          // precio mínimo Low
   double   close;        // precio de cierre Close
   long     tick_volume;  // volumen de ticks
   int      spread;       // spread
   long     real_volume;  // volumen de bolsa
  };
```

Variables globales del Terminal de Cliente

Durante la simulación, las variables globales del Terminal de Cliente se emulan también, pero no están relacionadas de ninguna manera con auténticas variables globales del terminal que se puede ver en el terminal utilizando el botón F3. Eso quiere decir que todas las operaciones con las variables globales del terminal durante la simulación se realizan fuera del mismo (en el agente de pruebas).

Cálculo de indicadores durante la simulación

En el modo de tiempo real los valores de los indicadores se calculan sobre cada tick. En el Probador se utiliza el modelo económico de cálculo de indicadores - los indicadores se recalculan justamente antes de que se arranque la ejecución del EA. Eso significa que el recálculo de valores de los indicadores se hace antes de la llamada a las funciones OnTick(), OnTrade() y OnTimer().

> No importa si hay o no la llamada al indicador en un manejador de eventos concreto. Todos los indicadores cuyos handles han sido creados por la función iCustom() o IndicatorCreate(), serán recalculados obligatoriamente antes de la llamada a la función del manejador de eventos.

De esta manera, durante la simulación en el modo "Todos los ticks" el cálculo de indicadores se hace antes de cada llamada a OnTick(). Si en el EA está activado el temporizador mediante la función EventSetTimer(), entonces los indicadores serán recalculados antes de cada llamada al handle OnTimer(). Por consecuencia, el tiempo de simulación puede aumentarse considerablemente sin en el EA se usa un indicador escrito de forma no óptima.

Carga del historial durante la simulación

El historial para el instrumento a probar se sincroniza y se descarga por el terminal desde el servidor comercial antes del inicio del proceso de simulación. En este caso, el terminal descarga desde el servidor comercial por primera vez todo el historial disponible para el instrumento a probar para luego ya no volver a este asunto. A continuación, se descargan sólo los datos nuevos.

El agente de pruebas recibe del Terminal de Cliente el historial para el instrumento a probar justamente después de que haya sido iniciado el proceso de simulación. Si durante el proceso de simulación se utilizan los datos de otros instrumentos (por ejemplo, si se trata de un EA de múltiples divisas), entonces en este caso el agente de pruebas solicita al Terminal de Cliente el historial necesario durante la primera invocación. Si los datos históricos se encuentran dentro del terminal, se pasan enseguida a los agentes de pruebas. Si no hay datos necesarios, el terminal los solicitará y descargará desde el servidor, y luego los pasará a los agentes de pruebas.

También se realiza el acceso a los instrumentos adicionales en el caso cuando se calcula el precio del tipo de cambio cruzado durante las operaciones de trading. Por ejemplo, durante la simulación de la estrategia sobre EURCHF con la moneda del depósito en dólares de los EE.UU. el agente de pruebas solicita al Terminal de Cliente el historial para EURUSD y USDCHF antes de procesar la primera operación de comercial, aunque la estrategia no supone la invocación directa a estos instrumentos financieros.

Antes de empezar a probar una estrategia de múltiples divisas, se recomienda descargar previamente todos los datos históricos necesarios al Terminal de Cliente. Esto permitirá evitar las demoras durante la simulación/optimización relacionadas con la descarga complementaria de datos que faltan. Por ejemplo, puede descargar el historial si abre los gráficos correspondientes y los desplaza hacia el inicio del historial. Puede encontrar un ejemplo de la descarga forzosa del historial al Terminal de Cliente en el apartado Organización de acceso a los datos de la documentación sobre MQL5.

Los agentes de pruebas en su lugar reciben el historial en forma comprimida desde terminal. Durante la simulación repetida el Probador ya no vuelve a descargar el historial desde el terminal, puesto que quedan los datos después del arranque anterior del Probador.

- El terminal descarga el historial desde el servidor comercial sólo una vez cuando el agente se dirige al terminal a por el historial para el símbolo a probar. El historial se descarga en forma comprimida con el fin de ahorrar el tráfico.
- Los ticks no se mandan por la red, sino se generan por los agentes de pruebas.

Simulación en múltiples divisas

El Probador permite llevar a cabo la simulación sobre el historial de estrategias que tradean utilizando varios instrumentos financieros. A estos EAs se les llaman condicionalmente de múltiples divisas, porque desde el principio en la plataformas anteriores la simulación se realizaba sólo para un instrumento financiero. Mientras que en el Probador del terminal MetaTrader 5 se puede simular el trading con todos los instrumentos disponibles.

El historial para los instrumentos utilizados se descarga por el Probador desde el **Terminal de Cliente** (¡no desde el servidor comercial!) de forma automática cuando se le llama al instrumento en cuestión por primera vez.

El agente de pruebas descarga sólo el historial que falta con pequeña reserva con el fin de asegurar los datos históricos necesarios para el cálculo de los indicadores en el momento de simulación. El volumen mínimo del historial a descargar desde el servidor comercial para los períodos D1 e inferiores es de un

año. De esta manera, si se inicia la simulación en el intervalo 2010.11.01-2010.12.01 (simulación en el intervalo de un mes) con el período M15 (cada barra es igual a 15 minutos), entonces se le solicitará al terminal el historial para el instrumento durante el año 2010 entero. Para los períodos Weekly será solicitado el historial de 100 barras, lo que supone aproximadamente dos años (hay 52 semanas en el año). Para la simulación sobre el período mensual Monthly el agente solicitará el historial de 8 años (12 meses * 8 años = 96 meses).

Si por alguna razón resulta imposible conseguir antes del inicio de la prueba el número de barras necesario para realizar la simulación, entonces la **fecha del inicio de la simulación será acercada automáticamente** hacia el presente para alcanzar esta reserva necesaria.

Durante la simulación se emula también la "Observación del Mercado" desde la cual se puede obtener la información sobre los instrumentos. Por defecto, al comienzo de la simulación la "Observación del Mercado" del Probador contiene sólo un símbolo, para el que ha sido iniciada la simulación. Todos los símbolos necesarios se conectan a la "Observación del Mercado" del Probador (¡no del Terminal!) de forma automática en cuanto se hace la llamada a ellos.

> Antes de empezar la simulación de un EA de múltiples divisas, hay que seleccionar los instrumentos necesarios para esta simulación en la "Observación del Mercado" del terminal y bajar los datos necesarios en la profundidad necesaria. Al llamar a un símbolo "ajeno" por primera vez, se ejecuta automáticamente la sincronización para este símbolo entre el agente de pruebas y el Terminal de Cliente. Un símbolo "ajeno" es aquél que se diferencia del símbolo sobre el que ha sido iniciada la simulación.

La llamada a los datos del símbolo ajeno se hace en las siguientes ocasiones:

- uso de las funciones de los indicadores técnicos y IndicatorCreate() sobre el par símbolo/período;
- llamada a la "Observación del Mercado" (Market Watch) para el símbolo ajeno:

1. SeriesInfoInteger
2. Bars
3. SymbolSelect
4. SymbolIsSynchronized
5. SymbolInfoDouble
6. SymbolInfoInteger
7. SymbolInfoString
8. SymbolInfoTick
9. SymbolInfoSessionQuote
10. SymbolInfoSessionTrade
11. MarketBookAdd
12. MarketBookGet

- llamada a las series temporales para el par símbolo/período utilizando las funciones:

1. CopyBuffer
2. CopyRates
3. CopyTime
4. CopyOpen

5. CopyHigh
6. CopyLow
7. CopyClose
8. CopyTickVolume
9. CopyRealVolume
10. CopySpread

En el momento cuando se hace la primera invocación a un símbolo ajeno, el proceso de simulación se detiene y se realiza la descarga adicional de datos históricos que faltan para el par símbolo/período desde el terminal al agente de pruebas. Al mismo tiempo se activa el proceso de generación de la secuencia de ticks para este símbolo.

Para cada instrumento se genera su propia secuencia de ticks de acuerdo con el modo de generación de ticks seleccionado. Aparte de eso, se puede solicitar el historial para los símbolos necesarios de forma explícita mediante la llamada a la función SymbolSelect() en el manejador OnInit(). En este caso el historial será descargado en el acto, antes que empiece a probar su EA.

De esta manera, para llevar a cabo la simulación de múltiples divisas en el Terminal de Cliente MetaTrader 5, no hace falta hacer ningunos esfuerzos adicionales. Bastará con abrir los gráficos de los instrumentos correspondientes en el Terminal de Cliente. El historial de los símbolos necesarios se descargará automáticamente desde el servidor comercial con la condición si dispone de estos datos.

Modelación de la hora en el Probador

Durante la simulación la hora local TimeLocal() siempre es iguala a la hora del servidor TimeTradeServer(). En su lugar, la hora del servidor siempre es iguala la hora que corresponde a la hora GMT - TimeGMT(). Así, durante la simulación todas estas funciones muestran la misma hora.

La falta de diferencia entre GMT, la hora local y de servidor en el Probador está hecha a propósito debido a que la conexión con el servidor no siempre puede ser permanente. Mientras que los resultados de la simulación tienen que ser iguales, independientemente de que si hay conexión o no. La información sobre la hora de servidor no se guarda de forma local, sino se coge en el servidor.

Objetos gráficos durante la simulación

La construcción de los objetos gráficos no se hace durante la simulación/optimización. De esta manera, el EA obtiene los valores cero cuando se dirige a las propiedades del objeto creado durante la simulación/optimización.

> Esta limitación no concierne a la simulación en modo visual.

Función OnTimer() en el Probador

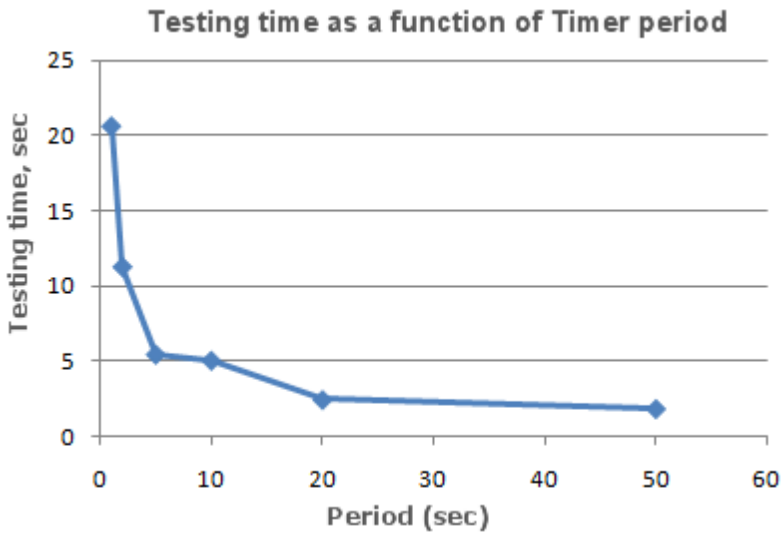
En MQL5 se puede procesar los eventos del temporizador. La llamada al manejador OnTimer() se realiza independientemente del modo de simulación. Eso significa que si la simulación ha sido iniciada en el modo "Sólo los precios de apertura" sobre el período H4 y dentro del EA está instalado un temporizador con la llamada cada segundo, entonces durante la apertura de cada barra H4 el manejador OnTick() será llamado una vez, y 14400 veces (3600 segundos * 4 horas) durante la barra será llamado el manejador OnTimer(). En cuánto va a aumentarse el tiempo de simulación depende de

la lógica del EA.

Hemos escrito un simple EA sin operaciones comerciales para comprobar la dependencia del tiempo de simulación de la periodicidad del temporizador.

```
//--- input parameters
input int      timer=1;              // valor del temporizador, segundos
input bool     timer_switch_on=true; // temporizador activado
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- iniciamos el temporizador si timer_switch_on==true
   if(timer_switch_on)
     {
      EventSetTimer(timer);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- detenemos el temporizador
   EventKillTimer();
  }
//+------------------------------------------------------------------+
//| Timer function                                                   |
//+------------------------------------------------------------------+
void OnTimer()
  {
//---
// no hacemos nada, el cuerpo del manejador está vacío
  }
//+------------------------------------------------------------------+
```

Se ha medido el tiempo de simulación con diferentes valores del parámetro timer (periodicidad del evento Timer). A base de los datos obtenidos se ha construido el gráfico de dependencia del tiempo de simulación T del valor de periodicidad Period.



Aquí se ve muy bien, cuanto más bajo sea el parámetro timer en el momento de inicialización del temporizador por la función EventSetTimer(timer), es menor el período (Period) entre las llamadas al manejador OnTimer(), y es mayor el tiempo de simulación T durante las mismas condiciones restantes.

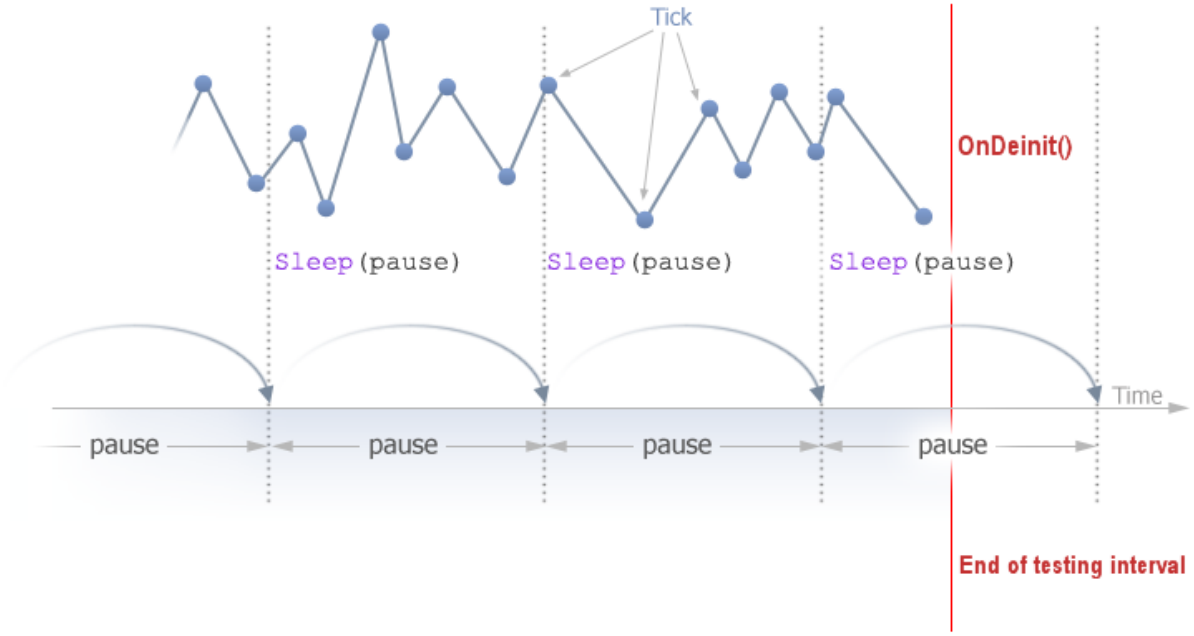
Función Sleep() en el Probador

La función Sleep() permite parar temporalmente la ejecución del programa mql5 en el EA o script durante el trabajo en el gráfico. Esto puede ser útil cuando se solicitan algunos datos que en el momento de la solicitud aún no están listos y hace falta esperar hasta que estén disponibles. Puede encontrar un ejemplo detallado del uso de la función Sleep() en el apartado Organización de acceso a los datos.

Pero en el Probador las llamadas a la función Sleep() no retrasan el proceso de simulación. Cuando se llama a la función Sleep(), "se reproducen" los ticks generados dentro del margen del retraso especificado, como resultado de lo cual pueden accionarse las órdenes pendientes, stops, etc. Después de la llamada a la función Sleep(), el tiempo modelado en el Probador se aumenta al intervalo especificado en el parámetro de la función Sleep.

Si como resultado de la ejecución de la función Sleep() el tiempo actual en el Probador ha salido fuera del período de simulación, verá el mensaje del error "ciclo infinito en Sleep". En caso de este error, los resultados de la simulación no se omiten. Todos los cálculos se realizan íntegramente (número de transacciones, reducciones, etc.).

La función Sleep() no va a funcionar en OnDeinit(), ya que tras su llamada el tiempo de simulación sí o sí saldrá fuera del intervalo de simulación.



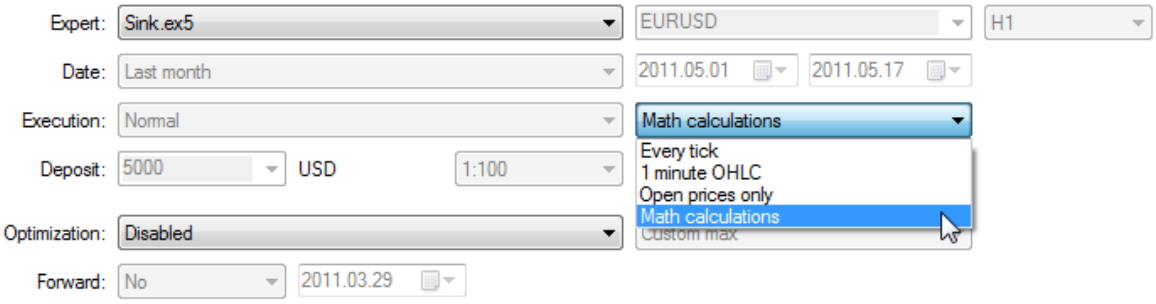
Función Print() en el Probador

Para aumentar la velocidad de respuesta y ahorrar el tráfico, la función Print() no funciona durante la simulación, ni para el uso de los agentes locales, ni tampoco para los agentes remotos o de la nube.

La excepción es el uso de la función Print() dentro del manejador OnInit(). Esto permite facilitar la búsqueda de las causas de los errores cuando éstos surjan.

Uso del Probador para las tareas de optimización en los cálculos matemáticos

En el terminal MetaTrader 5 se puede utilizar el Probador no sólo para probar las estrategias comerciales, sino también para los cálculos matemáticos. Para eso hay que seleccionar el modo correspondiente en las opciones:



Al elegir el modo "Cálculos matemáticos", será realizado un repaso en "vacío" del agente de simulación. El repaso en "vacío" significa que no se realizará la generación de los ticks ni tampoco va a cargarse el historial. Durante este repaso sólo serán llamadas las funciones OnInit(), OnTester() y OnDeinit().

Si la fecha de finalización de la prueba es menor o igual a la fecha de su inicio, esto también va a

significar la simulación en el modo "Cálculos matemáticos".

Durante el uso del Probador para la solución de tareas matemáticas, la descarga del historial y la generación de los ticks no se hace.

Una tarea matemática muy típica a resolver en el Probador del MetaTrader 5 es la búsqueda del extremo de la función de muchas variables. Para su solución hace falta:

- Colocar el bloque de cálculos del valor de la función de muchas variables en OnTester(), y devolver el valor calculado a través de return(valor_de_la_función);
- Pasar los parámetros de la función al área global del programa en forma de las variables input;

Compilamos el EA, abrimos la ventana "Probador". En la pestaña "Parámetros de entrada" marcamos los parámetros de entrada necesarios y establecemos para ellos los límites en el espacio de los valores y el paso para el repaso.

Seleccionamos el tipo de optimización: "Lenta" (repaso completo de parámetros) o "Rápida" (algoritmo genético). Es mejor seleccionar la optimización rápida para la simple búsqueda del extremo de la función. Pero si hace falta calcular los valores en todo el espacio de las variables, mejor conviene la optimización lenta.

Seleccionamos el modo "Cálculos matemáticos" e iniciamos el proceso de optimización haciendo clic en el botón "Empezar". Hay que recordar que durante la optimización siempre se busca el máximo local del valor de la función OnTester. Para la búsqueda del mínimo local se puede devolver de la función OnTester el valor inverso al valor calculado de la función:

```
return(1/valor_de_la_función);
```

En este caso Usted mismo debe comprobar que el valor_de_la_función no sea igual a cero, ya que de lo contrario se puede recibir un error crítico de división por cero. Hay otra opción que es más conveniente y que no altera los resultados de la optimización, ha sido ofrecido por los lectores del artículo:

```
return(-valor_de_la_función);
```

Aquí no hace falta comprobar si el valor_de_la_función es igual a cero, y la misma superficie de los resultados de la optimización en representación 3D tiene la misma forma pero reflejada de espejo respecto a la inicial.

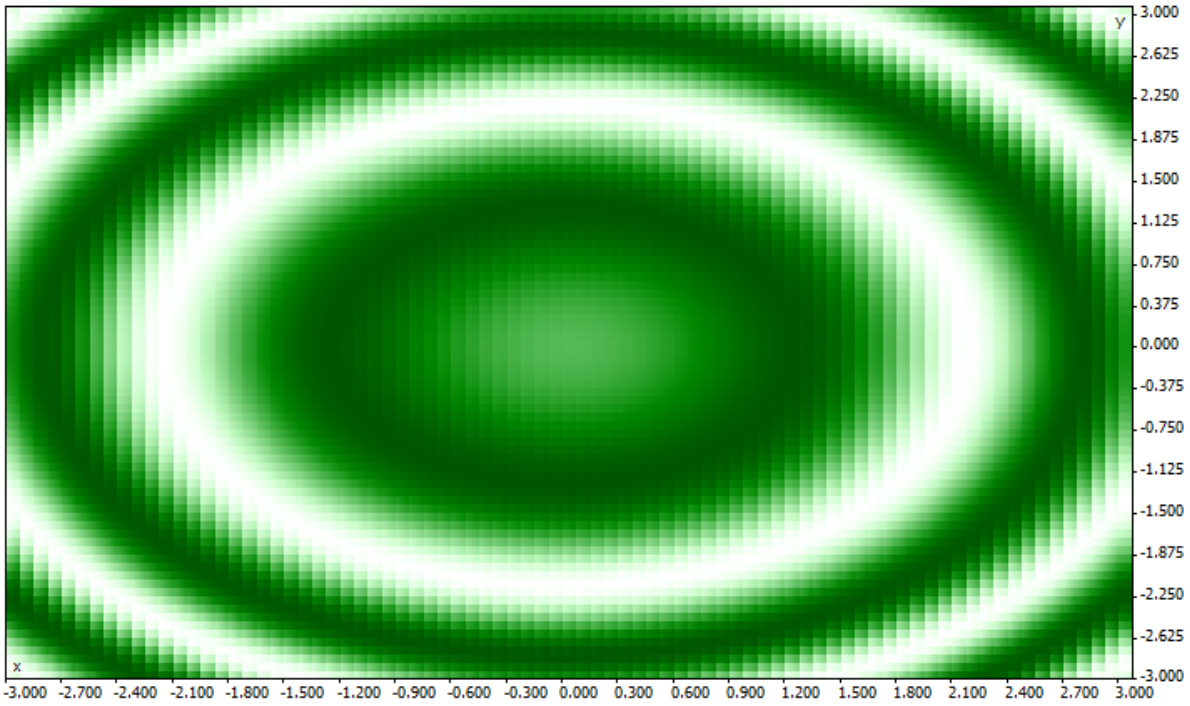
Como ejemplo vamos a coger la función sink():

$$sink(x,y) = \sin(x^2 + y^2)$$

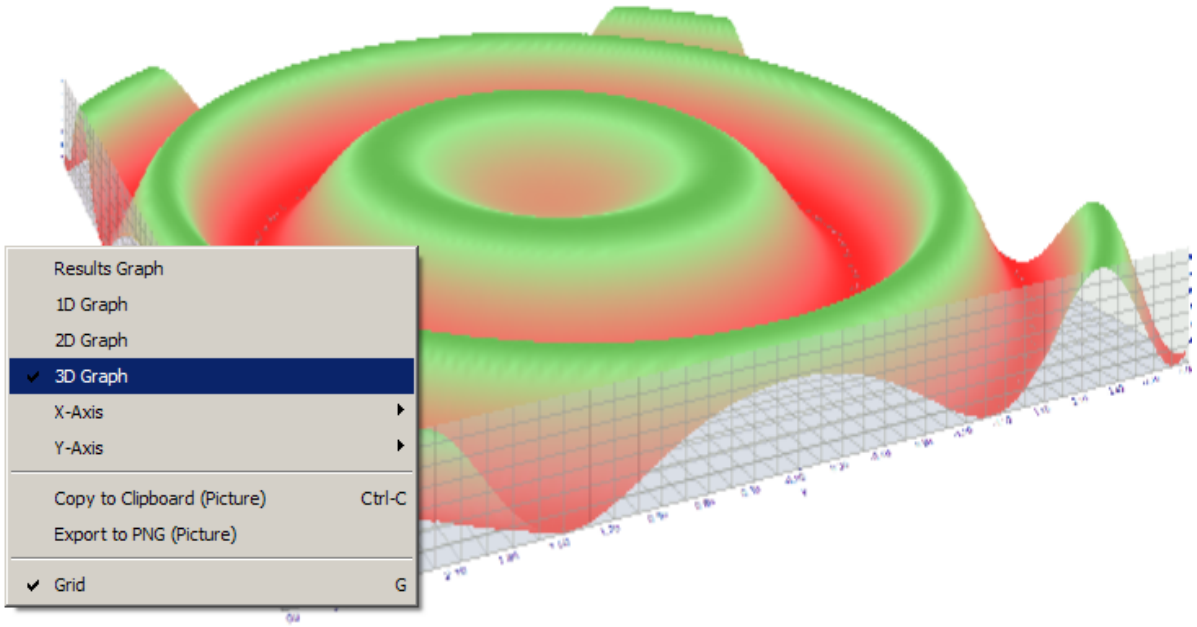
El código del EA para la búsqueda del extremo de esta función vamos a colocar en OnTester():

```
//+------------------------------------------------------------------+
//|                                                        Sink.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- input parameters
input double   x=-3.0; // start=-3, step=0.05, stop=3
input double   y=-3.0; // start=-3, step=0.05, stop=3
//+------------------------------------------------------------------+
//| Tester function                                                  |
//+------------------------------------------------------------------+
double OnTester()
  {
//---
   double sink=MathSin(x*x+y*y);
//---
   return(sink);
  }
//+------------------------------------------------------------------+
```

Hagamos la optimización y mostremos los resultados de la optimización como un gráfico 2D.



Cuanto mejor sea el valor para el par de parámetros establecido (x, y), más denso será el color. Tal como se esperaba partiendo de la vista de la fórmula de la función sink(), sus valores forman unos círculos concéntricos con el centro en el punto (0,0). Para la función sink() no existe un extremos absoluto. Esto se ve muy bien cuando vemos los resultados de optimización en el modo 3D:



Sincronización de las barras durante la simulación en el modo "Sólo precios de apertura"

El Probador en el terminal MetaTrader 5 permite probar también así llamados Asesores Expertos "de múltiples divisas". Un EA de múltiples divisas es un Asesor Experto que opera con dos o más símbolos.

La simulación de las estrategias que tradean con varios instrumentos impone al Probador unos requerimientos técnicos adicionales:

- generación de ticks para estos instrumentos;
- cálculo de valores de los indicadores para estos instrumentos;
- cálculo de requerimientos del margen para estos instrumentos;
- sincronización de secuencias de ticks generadas para todos los instrumentos con los que se tradea.

El Probador genera y reproduce una secuencia de ticks para cada instrumento en función del modo de trading seleccionado. En este caso la nueva barraen cada instrumento se abre independientemente de cómo se ha abierto la barra en otro instrumento. Eso significa que durante la simulación de un EA de múltiples divisas puede surgir la situación (suele pasar con bastante frecuencia) cuando en un instrumento la barra ya se ha abierto y en el otro todavía no. De esta manera, durante la simulación pasa lo mismo que pasa en la vida real.

Esta auténtica modelación del desarrollo del historial en el Probador no causa preguntas hasta que utilizamos los modos de simulación "Todos los ticks" y "1 minute OHLC". Durante el uso de estos modos, dentro de los límites de una vela se genera una cantidad de ticks suficiente para esperar el momento de sincronización de las barras de diferentes símbolos. ¿Pero cómo vamos a probar las estrategias de múltiples divisas en el modo "Sólo precios de apertura" si se requiere la sincronización obligatoria de las barras para los instrumentos en los que tradeamos? Pues, en este modo el EA es llamado sólo en el tick que correspondiente a la hora de apretura de la barra.

Lo explicaremos en un ejemplo: si probamos nuestro EA sobre el símbolo EURUSD, y en EURUSD ha sido abierta una nueva vela, será muy fácil enterarnos de eso. Es que durante la simulación en el modo "Sólo precios de apertura", el evento NewTick corresponde al momento de apertura de la barra

en el período de la prueba. Pero no existe garantía alguna de que la nueva vela se ha abierto para el símbolo GBPUSD que se utiliza en el EA.

En condiciones normales bastará con finalizar el trabajo de la función OnTick() y comprobar la aparición de la nueva barra para GBPUSD durante el siguiente tick. Pero durante la simulación en el modo "Sólo precios de apertura" no habrá ningún otro tick, y a lo mejor puede formarse la impresión que este modo no vale para probar los EAs de múltiples divisas. Pero eso no es así. No olvide que el Probador en MetaTrader 5 se comporta igual como en la vida real. Se puede esperar el momento cuando para el otro símbolo se abrirá una nueva barra mediante la función Sleep()!

Este es código del EA Synchronize_Bars_Use_Sleep.mq5 que muestra el ejemplo de sincronización de las barras durante la simulación en el modo "Sólo precios de apertura":

```
//+------------------------------------------------------------------+
//|                                  Synchronize_Bars_Use_Sleep.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- input parameters
input string   other_symbol="USDJPY";
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- confrontamos el símbolo actual
   if(_Symbol==other_symbol)
     {
      PrintFormat("¡Hace falta especificar otro símbolo o iniciar la simulación sobre
      //--- finalizamos la prueba forzosamente
      return(INIT_PARAMETERS_INCORRECT);
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- variable estática para guardar la hora de apertura de la última barra
   static datetime last_bar_time=0;
//---  indicio de que la hora de apertura de la última barra de diferentes símbolos e
   static bool synchonized=false;
//--- si la variable estática aún no está inicializada
   if(last_bar_time==0)
     {
      //--- es la primera llamada, apuntaos la hora de apertura y salimos
      last_bar_time=(datetime)SeriesInfoInteger(_Symbol,Period(),SERIES_LASTBAR_DATE)
      PrintFormat("Hemos inicializado la variable last_bar_time con el valor %s",Time
     }
//--- obtenemos la hora de apertura de la última barra para nuestro símbolo
   datetime curr_time=(datetime)SeriesInfoInteger(Symbol(),Period(),SERIES_LASTBAR_DA
//--- si la hora de apertura de la barra actual no coincide con la que se guarda en l
   if(curr_time!=last_bar_time)
     {
      //--- recordamos la hora de apertura de la nueva barra en la variable estática
      last_bar_time=curr_time;
      //--- la sincronización ha sido violada, mostramos la bandera false
      synchonized=false;
      //--- mostramos el mensaje sobre este evento
      PrintFormat("Para el símbolo %s se ha abierto nueva barra a las %s",_Symbol,Tim
     }
//--- aquí vamos a guardar la hora de apertura de la barras para el símbolo ajeno
   datetime other_time;
//--- ciclo, hasta que la hora de apertura de la última barra para el otro símbolo co
   while(!(curr_time==(other_time=(datetime)SeriesInfoInteger(other_symbol,Period(),SI
     {
      PrintFormat("Esperaremos 5 segundos..");
      //--- esperaremos 5 segundos y volveremos a solicitar SeriesInfoInteger(other_s
      Sleep(5000);
```

```
      }
//--- la hora de apertura de la barra ahora es igual para los dos símbolos
   synchonized=true;
   PrintFormat("La hora de apertura de la última barra para nuestro símbolo %s: %s",_
   PrintFormat("La hora de apertura de la última barra para el símbolo %s: %s",other_
//--- TimeCurrent() no vale, utilizamos TimeTradeServer() para
   Print("Las barras han sido sincronizadas a las ",TimeToString(TimeTradeServer(),TI
  }
//+------------------------------------------------------------------+
```

Fíjense en la última línea del EA que nos muestra la hora actual a la que ha sido determinado el hecho de sincronización:

```
   Print("Las barras han sido sincronizadas a las ",TimeToString(TimeTradeServer(),TI
```

Para mostrar la hora actual hemos utilizado la función TimeTradeServer(), en vez de la TimeCurrent(). Es que la función TimeCurrent() devuelve la hora del último tick que no se ha cambiado de ninguna manera tras el uso de Sleep(). Inicie el EA en el modo "Sólo precios de apertura" y verá los mensajes sobre la sincronización de las barras.

Core 1	2010.12.01 20:00:05	The bars are synchronized at 2010.12.01 20:00:05
Core 1	2010.12.01 20:00:05	Open bar time of the chart symbol USDJPY: 2010.12.01 20:00
Core 1	2010.12.01 20:00:05	A new bar has appeared on symbol EURUSD: 2010.12.01 20:00
Core 1	2010.12.01 20:00:00	Waiting 5 seconds..
Core 1	2010.12.01 20:00:05	A new bar has appeared on symbol EURUSD: 2010.12.01 20:00
Core 1	2010.12.01 16:00:05	The bars are synchronized at 2010.12.01 16:00:05

Utilice la función TimeTradeServer() en lugar de la TimeCurrent() si necesita obtener la hora de servidor actual y no la hora de llegada del último tick.

Hay otra forma de sincronizar las barras - utilizando el temporizador. El ejemplo de tal EA Synchronize_Bars_Use_OnTimer.mq5 se adjunta al artículo.

Función IndicatorRelease() en el Probador

Después de la finalización de la simulación, se abre automáticamente el gráfico del instrumento en el cual se muestran las transacciones realizadas y los indicadores que se han utilizado en el EA. Esto ayuda comprobar de forma visual los momentos de entrada y salida, así como compararlos con los valores de los indicadores.

> **Importante:** los indicadores mostrados en el gráfico abierto automáticamente tras finalizarse la simulación se calculan de nuevo ya después de completarse la prueba. Incluso si estos indicadores han sido utilizados en el EA a probar.

Pero en algunas ocasiones el programador puede necesitar ocultar la información sobre los indicadores utilizados en el algoritmo de trading. Por ejemplo, el código del EA se alquila o se vende como archivo ejecutable sin proporcionar el código fuente. Para este propósito valdrá la función IndicatorRelease().

Si en la carpeta /profiles/templates del Terminal de Cliente hay una plantilla que se llama tester.tpl, precisamente esta plantilla será aplicada al gráfico que se abre. Si no hay esta plantilla, se aplica la plantilla predeterminada (default.tpl).

Desde el principio la función IndicatorRelease() está destinada para liberar la parte de cálculo del indicador, en caso de que ya no es necesario. Esto permite ahorrar la memoria y los recursos de la CPU, porque cada tick activa el cálculo del indicador. Su segundo cometido consiste en prohibir la visualización del indicador en el gráfico de simulación tras finalizarse el repaso único.

Para prohibir la visualización del indicador en el gráfico después de la simulación, invoque la función IndicatorRelease() con el handle del indicador en el manejador OnDeinit(). La función OnDeinit() siempre se invoca después de la finalización y antes de visualización del gráfico de simulación.

```
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//---
   bool hidden=IndicatorRelease(handle_ind);
   if(hidden) Print("IndicatorRelease() ejecutada con éxito");
   else Print("IndicatorRelease() ha devuelto false. Código del error ",GetLastError(
  }
```

Para prohibir la visualización del indicador en el gráfico después de la simulación, utilice la función IndicatorRelease() en el manejador OnDeinit().

Procesamiento de eventos en el Probador

La presencia del manejador OnTick() en el EA no es obligatoria para que se pueda pobarlo sobre los datos históricos en el Probador del terminal MetaTrader 5. Será suficiente que en el EA haya por lo menos una función-manejador listadas más abajo:

- OnTick() - manejador del evento de la llegada de un nuevo tick;
- OnTrade() - manejador de un evento comercial;
- OnTimer() - manejador del evento de la llegada de una señal de temporizador;
- OnChartEvent() - manejador de los eventos del usuario.

Durante la simulación, en el Probador se puede procesar los eventos de usuario utilizando la función OnChartEvent(), pero en los indicadores esta función no se invoca en el Probador. Incluso si el indicador dispone del manejador OnChartEvent() y este indicador se utiliza en el EA, este indicador no va a recibir ningunos eventos personalizados.

Durante la simulación el indicador puede generar los eventos personalizados utilizando la función EventChartCustom(), y el EA puede procesar este evento en OnChartEvent().

Aparte de los eventos arriba mencionados en el Probador de Estrategias se generan los eventos especiales relacionados con el proceso de simulación y optimización:

- Tester - este evento se genera cuando se finaliza la simulación del EA a base de los datos históricos. El procesamiento del evento Tester se hace por la función OnTester(). Se puede utilizar esta función sólo en los EAs durante la simulación, y en primer lugar está destinada para calcular un valor que se utiliza como criterio Custom max durante la optimización genética de los parámetros de entrada.
- TesterInit - este evento se genera cuando se inicia el proceso de optimización en el Probador de Estrategias antes del primer repaso. El procesamiento del evento TesterInit se realiza por la función OnTesterInit(). El EA que dispone de este manejador se carga automáticamente, al iniciarse

la optimización, en un gráfico nuevo del terminal con el símbolo y período especificados en el Probador, y recibe el evento TesterInit. La función está destinada para inicializar el EA antes del inicio de la optimización para el posterior procesamiento de los resultados de la optimización.

- TesterPass - este evento se genera cuando llega un nuevo frame de datos. El procesamiento del evento TesterPass se realiza por la función OnTesterPass(). El EA con este manejador se carga automáticamente en un gráfico nuevo del terminal con el símbolo/período especificados para la simulación, y recibe durante la optimización el evento TesterPass cuando llegue un frame. La función está destinada para el procesamiento dinámico de los resultados de la optimización directamente "al vuelo", sin esperar su finalización. La agregación de los frames se realiza por la función FrameAdd(), que puede ser invocada cuando se finaliza el repaso único en el manejador OnTester().
- TesterDeinit - este evento se genera cuando se termina el proceso de optimización del EA en el Probador de Estrategias. El procesamiento del evento TesterDeinit se realiza por la función OnTesterDeinit(). El EA con este manejador se carga automáticamente en el gráfico al iniciarse la optimización y recibe el evento TesterDeinit tras su finalización. Esta función está destinada para el procesamiento final de todos los resultados de la optimización.

Agentes de pruebas

En el Terminal de Cliente MetaTrader 5 la simulación se realiza utilizando los agentes de pruebas. Los agentes locales se crean y se conectan de forma automática. Por defecto, el número de los agentes locales corresponde al número de núcleos que tiene el ordenador.

Cada agente de pruebas dispone de su propia copia de variables globales que no está relacionada de ninguna manera con el Terminal de Cliente. El mismo terminal desempeña el papel del operador que reparte las tareas para los agentes locales y remotos. Después de ejecutar la tarea de turno relacionada con la simulación de un EA con parámetros establecidos, el agente devuelve el resultado al terminal. Durante la prueba única se utiliza sólo un agente.

El agente guarda el historial que recibe del terminal en las carpetas separadas que llevan el nombre del instrumento. Es decir, el historial para EURUSD se guarda en la carpeta con el nombre EURUSD. Aparte de eso el historial de los instrumentos se separa según las fuentes. La estructura de almacenamiento del historial es la siguiente:

```
carpeta_del_probador\Agent-IPaddress-Port\bases\nombre_de_la_fuente\history\nombre_de
```

Por ejemplo, el historial para EURUSD del servidor MetaQuotes-Demo se puede guardar en la carpeta_del_probador\Agent-127.0.0.1-3000\bases\MetaQuotes-Demo\EURUSD.

Después de haberse finalizado el proceso de simulación, el agente local se encuentra durante cinco minutos en el modo de espera de la siguiente tarea para no perder tiempo con el arranque en caso de las siguientes llamadas. Y sólo transcurrido este plazo de espera, el agente local finaliza su trabajo y se descarga de la memoria del ordenador.

En caso de la finalización anticipada de la simulación por parte del usuario (el botón "Cancelar"), así como en caso del cierre del terminal, todos los agentes locales finalizan su trabajo y se descargan de la memoria del ordenador.

Intercambio de datos entre el Terminal y el Agente

Cuando se inicia el proceso de simulación, el terminal se prepara para enviar al agente unos bloques

de parámetros:

- Parámetros de entrada de simulación (modo de modelación, intervalo de simulación, instrumento, criterio de optimización, etc.)
- Lista de instrumentos seleccionados en "Observación del Mercado"
- Especificación del instrumento a probar (tamaño del contrato, desviaciones permitidas del mercado para colocar StopLoss y Takeprofit, etc.)
- EA a probar y los valores de sus parámetros de entrada
- Información sobre los archivos adicionales (bibliotecas, indicadores, archivos de datos - #property tester_...)

tester_indicator	string	Nombre del indicador personalizado en el formato "nombre_del_indicador.ex5". Los indicadores necesarios para la simulación se determinan automáticamente desde la llamada de la función iCustom(), si el parámetro correspondiente ha sido establecido con una constante literal. Para los demás casos (el uso de la función IndicatorCreate() o el uso de una cadena no constante en el parámetro que establece el nombre del indicador) hace falta esta propiedad
tester_file	string	Nombre del archivo para el Probador con extensión, encerrado entre dobles comillas (como constante literal). El archivo especificado será pasado al Probador. Siempre hay que especificar los archivos de entrada para la simulación, en caso de que haya necesidad de ellos
tester_library	string	Nombre de la biblioteca con extensión encerrado entre dobles comillas. Una biblioteca puede tener la extensión dll o ex5. Las bibliotecas necesarias para la simulación se determinan automáticamente. Sin embargo, si alguna biblioteca se utiliza por un indicador

		personalizado, esta propiedad es necesaria

Para cada bloque de parámetros se crea una huella digital en forma del hash MD5 que se envía al agente. El hash MD5 es único para cada conjunto de datos, siendo su volumen muchas veces menor que el volumen de información a base de la cual éste ha sido calculado.

El agente recibe los hashs de los bloques y los compara con los que ya tiene almacenados. Si el agente no dispone de la huella de este bloque de parámetros, o el hash recibido se diferencia del existente, el agente solicita el bloque de parámetros en sí. De esta manera, se reduce el volumen de tráfico entre el terminal y el agente.

Después de realizar la simulación, el agente devuelve al terminal todos los resultados de la prueba que se muestran en las pestañas "Resultados de simulación" y "Resultados de optimización": beneficio obtenido, número de transacciones, ratio de Sharpe, resultado de la función OnTester(), etc.

Durante la optimización, el terminal reparte entre los agentes las tareas para realizar la prueba utilizando pequeños paquetes de datos. Cada paquete contiene unas cuantas tareas (cada tarea supone una prueba única con el conjunto de parámetros de entrada). Esto reduce el tiempo de intercambio entre el terminal y el agente.

Los agentes nunca guardan en el disco duro los archivos EX5 recibidos del terminal (EA, indicadores, bibliotecas, etc.) por motivos de seguridad. Se hace para que no se pueda utilizar los datos recibidos en el ordenador con el agente instalado. Todos los demás datos, incluyendo DLL, se guardan en la zona protegida (sandbox). En los agentes remotos no se puede probar los EAs con el uso de las DLL.

El terminal deposita los resultados de simulación en una caché de resultados especial (caché resultante) para un acceso rápido a ellos cuando surja esta necesidad. Para cada conjunto de parámetros el terminal busca en la caché resultante los resultados ya listos de los arranques anteriores con el fin de evitar los arranques repetitivos. Si el resultado con este conjunto de parámetros no ha sido encontrado, el agente recibe la orden para empezar la prueba.

Todo el tráfico entre el terminal y el agente se codifica.

> Los ticks no se mandan por la red, sino se generan por los agentes de pruebas.

Uso de la carpeta compartida de todos los Terminales de Cliente

Todos los agentes de pruebas están aislados uno del otro y del Terminal de Cliente también: cada agente tiene su propia carpeta donde se guardan todos los logs del agente. Además de eso, durante la simulación todas las operaciones con los archivos se hacen en la carpeta **nombre_del_agente/MQL5/Files**. No obstante, se puede organizar la interacción entre los agentes locales y el terminal a través de la carpeta compartida de todos los terminales de cliente si durante la apertura del archivo indicamos la bandera FILE_COMMON:

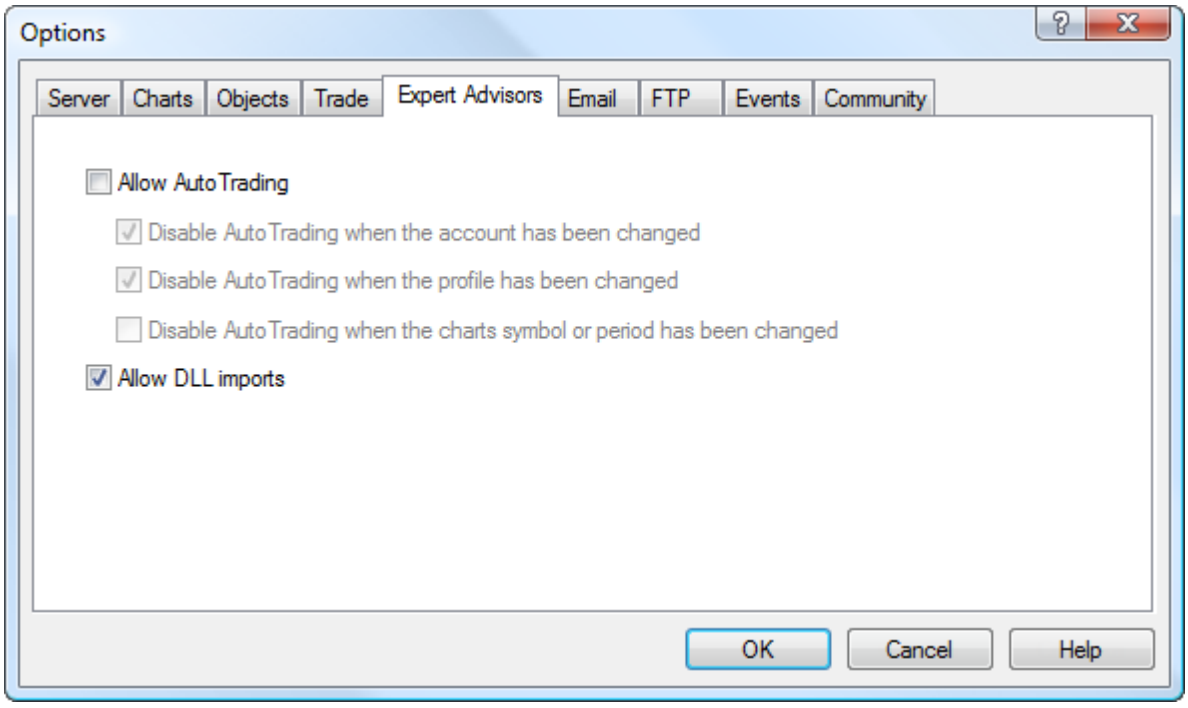
```
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
```

```
//--- carpeta compartida de todos los Terminales de Cliente
   common_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
//--- mostraremos el nombre de esta carpeta
   PrintFormat("Abrimos el archivo en la carpeta compartida de todos los Terminales d
//--- abrimos el archivo en la carpeta compartida (tenemos la bandera FILE_COMMON)
   handle=FileOpen(filename,FILE_WRITE|FILE_READ|FILE_COMMON);
  ... siguientes acciones
//---
   return(INIT_SUCCEEDED);
  }
```

Uso de la DLL

Para acelerar el proceso de la simulación, se puede utilizar no sólo los agentes locales, sino también los agentes remotos. Pero hay ciertas limitaciones para los agentes remotos. En primer lugar, los agentes remotos no muestran en sus logs los resultados de ejecución de la función Print(), los mensajes sobre la apertura y el cierre de posiciones. En el log se muestra el mínimo de información para que los EAs escritos de forma incorrecta no sobrellenen con sus mensajes el disco duro del ordenador en el que trabaja el agente remoto.

La segunda limitación consiste en prohibición del uso de DLL durante la simulación de los EAs. Las llamadas a las bibliotecas DLL están completamente prohibidas en los agentes remotos por motivos de seguridad. Para los agentes locales las llamadas dll dentro de los EAs que se prueban están permitidas sólo en el caso si el permiso correspondiente ha sido concedido por medio de la opción "Permitir importación de DLL".



Importante: si utiliza los EAs (scripts, indicadores) recibidos desde fuera que exigen que les permita las llamadas a DLL, Usted debe comprender todo el riesgo que toma sobre sí en caso de permitir el uso de esta opción en los ajustes del terminal. Y eso no depende de forma del uso del EA -para la simulación o para su arranque en el gráfico.

Variables predefinidas

Para cada programa MQL5 en ejecución se soporta una serie de variables predefinidas. Éstas reflejan el estado de gráfico de precios corriente en el momento cuando el programa (Experto, script o indicador personalizado) se inicie.

El terminal de usuario establece los valores para las variables predefinidas antes de que se inicie el programa mql5. Las variables predefinidas son constantes y no pueden ser cambiadas desde el programa mql5, salvo la variable _LastError, la cual puede ser anulada por la función ResetLastError.

Variable	Valor
_Digits	Número de dígitos después de la coma decimal
_Point	Tamaño del punto del instrumento corriente en divisa de cotización
_LastError	Valor del último error
_Period	Valor del período de gráfico corriente
_RandomSeed	Estado actual del generador de números pseudoaleatorios
_StopFlag	Bandera de detención del programa
_Symbol	Nombre del símbolo del gráfico corriente
_UninitReason	Código de la causa de deinicialización

Las bibliotecas utilizan las variables del programa que las ha invocado.

int _Digits

En la variable _Digits se guarda el número de dígitos después de la coma decimal que determina la precisión de cálculo del precio del símbolo del gráfico corriente.

También se puede usar la función Digits().

double _Point

En la variable _Point se guarda el tamaño del punto de instrumento corriente en divisa de cotización.

También se puede usar la función Point().

int _LastError

En la variable _LastError se guarda el valor del último error ocurrido durante la ejecución del programa mql5. La función ResetLastError() puede reiniciar este valor con el 0.

También se puede usar la función GetLastError() para obtener el código del error.

int _Period

En la variable _Period se guarda el valor del período de gráfico corriente.

También se puede usar la función Period().

Véase también

PeriodSeconds, Períodos de gráficos, Fecha y hora, Visibilidad de objetos

_RandomSeed

Una variable para guardar el estado actual durante la generación de números pseudoaleatorios enteros. _RandomSeed cambia su valor al llamar a MathRand(). Para establecer el estado inicial necesario, utilice MathSrand().

Un número aleatorio ***x*** que obtiene la función MathRand() se calcula durante cada llamada de la siguiente manera:

```
x=_RandomSeed*214013+2531011;
_RandomSeed=x;
x=(x>>16)&0x7FFF;
```

Véase también

MathRand(), MathSrand(), Tipos enteros

bool _StopFlag

En la variable _StopFlag se guarda la bandera de detención del programa mql5. Cuando el terminal de cliente intenta detener el programa, en esta variable se inscribe el valor true.

Para comprobar el valor de la bandera _StopFlag también se puede usar la función IsStopped().

string _Symbol

En la variable _Symbol se guarda el nombre del símbolo del gráfico corriente.

También se puede usar la función Symbol().

int _UninitReason

En la variable _UninitReason se guarda el código de la causa de deinicialización del programa.

Suelen obtener el código de la causa de deinicialización usando la función UninitializeReason().

Funciones comunes

Funciones generales que no han entrado en ninguno de los grupos especializados.

Función	Acción
Alert	Muestra el mensaje en una ventana separada
CheckPointer	Devuelve el tipo del puntero a objeto
Comment	Muestra el mensaje en la esquina superior izquierda del gráfico de precios
DebugBreak	Punto programado de interrupción en depuración
ExpertRemove	Detiene el trabajo del Asesor Experto y lo descarga del gráfico
GetPointer	Devuelve el puntero a objeto
GetTickCount	Devuelve la cantidad de millisegundos pasados desde el momento del arranque del sistema
MessageBox	Crea y visualiza la ventana de mensajes, además de gestionarlo
PeriodSeconds	Devuelve la cantidad de segundos en el período
PlaySound	Reproduce un archivo de audio
Print	Visualiza un mensaje en el registro
PrintFormat	Formatea e imprime juego de símbolos y valores en un registro histórico de acuerdo con el formato establecido
ResetLastError	Pone el valor de variable predefinida _LastError a cero
ResourceSave	Guarda el recurso en el archivo especificado
SendFTP	Envía un archivo a la dirección especificada en la ventana de configuraciones, en la pestaña "Edición"
SendMail	Envía una carta electrónica a la dirección especificada en la ventana de configuraciones, en la pestaña "Edición"
Sleep	Suspende la ejecución del Asesor Experto en curso o del script por un intervalo determinado
TerminalClose	Envía al terminal el comando de finalizar el trabajo
TesterStatistics	La función devuelve el valor del indicador estadístico especificado que ha sido calculado a base de los resultados de simulación

ZeroMemory	Reinicializa la variable pasada por referencia. La variable puede ser de cualquier tipo, salvo las clases y estructuras que contienen constructores.

Alert

Visualiza la ventana de diálogo que contiene los datos de usuario.

```
void  Alert(
   argument,      // el primer valor
   ...            // siguientes valores
   );
```

Parámetros

argument

[in] Cualquieres valores separados por comas. Para separar la información mostrada en varias líneas se puede usar el símbolo de avance de líneas "\n" o "\r\n". El número de parámetros no puede superar 64.

Valor devuelto

No hay valor devuelto.

Nota

No se puede pasar los arrays a la función Alert(). Los arrays deben visualizarse elemento por elemento. Los datos del tipo double se visualizan con 8 dígitos decimales después del punto, los del tipo float - con 5 dígitos decimales después del punto. Para visualizar los números reales con otra precisión o en formato científico hace falta usar la función DoubleToString().

Los datos del tipo bool se visualizan como cadenas "true" o "false". Las fechas se visualizan como YYYY.MM.DD HH:MI:SS. Para conseguir otro formato de fecha hay que usar la función TimeToString (). Los datos del tipo color se visualizan como cadena R,G,B, o usando el nombre del color si está presente en el juego de colores.

CheckPointer

Devuelve el tipo de puntero a objeto.

```
ENUM_POINTER_TYPE  CheckPointer(
   object* anyobject      // puntero a objeto
   );
```

Parámetros

anyobject

[in] Puntero a objeto.

Valor devuelto

Devuelve el valor de enumeración ENUM_POINTER_TYPE.

Nota

El intento de llamar a un puntero incorrecto lleva a la terminación crítica del programa. Por eso es necesario usar la función CheckPointer antes de usar un puntero. Un puntero puede ser incorrecto en las siguientes ocasiones:

- el puntero es igual a NULL;
- si el objeto ha sido eliminado por el operador delete.

Esta función puede ser utilizada para comprobar la validez del puntero. Un valor diferente a cero garantiza que el puntero puede ser utilizado para acceder.

Ejemplo:

```
//+------------------------------------------------------------------+
//|  eliminación de la lista mediante eliminación de sus elementos   |
//+------------------------------------------------------------------+
void CMyList::Destroy()
  {
//--- puntero auxiliar para trabajar en el ciclo
   CItem* item;
//--- pasamos por el ciclo e intentamos eliminar los punteros dinámicos
   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 deleted");
     }
//---
  }
```

Véase también

Punteros a objetos, Comprobación de punteros a objetos, Operador de eliminación de objeto delete

Comment

Visualiza el comentario definido por el usuario en la esquina superior izquierda del gráfico.

```
void  Comment(
   argument,      // el primer valor
   ...            // siguientes valores
   );
```

Parámetros

...

[in] Cualquieres valores separados por comas. Para separar la información mostrada en varias líneas se puede usar el símbolo de avance de líneas "\n" o "\r\n". El número de parámetros no puede superar 64. La longitud total del mensaje a mostrar (inclusive los símbolos auxiliares invisibles) no podrá superar 2045 símbolos (los que sobran van a ser cortados a la hora de ser mostrados).

Valor devuelto

No hay valor devuelto

Nota

No se puede pasar los arrays a la función Comment(). Los arrays tienen que imprimirse elemento por elemento.

Los datos del tipo double se visualizan con la precisión de hasta 16 dígitos decimales después del punto, además, los datos pueden ser visualizados en el formato tradicional o científico, dependiendo de cuál de ellos va a ser más compacto. Los datos del tipo float se visualizan con 5 dígitos decimales después del punto. Para visualizar los números reales con otra precisión o en un formato explícitamente especificado hay que usar la función DoubleToString().

Los datos del tipo bool se visualizan como cadenas "true" o "false". Las fechas se visualizan como YYYY.MM.DD HH:MI:SS. Para conseguir otro formato de fecha hay que usar la función TimeToString (). Los datos del tipo color se visualizan como cadena R,G,B, o usando el nombre del color si está presente en el juego de colores.

Ejemplo:

```
void OnTick()
  {
//---
   double Ask,Bid;
   int Spread;
   Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   Spread=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD);
//--- Mostremos los valores en tres líneas
   Comment(StringFormat("Mostramos precios\nAsk = %G\nBid = %G\nSpread = %d",Ask,Bid,
  }
```

Véase también

ChartSetString, ChartGetString

DebugBreak

Es el punto programado de interrupción en depuración.

```
void  DebugBreak();
```

Valor devuelto

No hay valor devuelto.

Nota

La ejecución de un programa mql5 se interrumpe sólo si el programa está inicializado en modo de depuración. Se puede utilizar esta función para ver los valores de las variables y/o la siguiente ejecución paso a paso.

ExpertRemove

Detiene el trabajo del Asesor Experto y lo descarga del gráfico.

```
void  ExpertRemove();
```

Valor devuelto

No hay valor devuelto.

Nota

El Asesor Experto no se detiene inmediatamente si se llama a la función ExpertRemove(), únicamente se activa la bandera para detener la ejecución del Asesor Experto. Es decir, el Asesor Experto no va a procesar ninguno de los siguientes eventos, se invocará la función OnDeinit() y El Asesor Experto será descargado y borrado del gráfico.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                            Test_ExpertRemove.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
input int ticks_to_close=20;// número de ticks antes de descargar el Asesor Experto
//+------------------------------------------------------------------+
//| 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("\nHasta la descarga del Asesor Experto ",__FILE__," quedan",
           (ticks_to_close-tick_counter)," ticks");
//--- hasta
   if(tick_counter>=ticks_to_close)
     {
      ExpertRemove();
      Print(TimeCurrent(),": ",__FUNCTION__," Asesor Experto será descargado");
     }
   Print("tick_counter = ",tick_counter);
//---
  }
//+------------------------------------------------------------------+
```

Véase también

Funcionamiento de programas, Eventos del terminal de cliente

GetPointer

Devuelve el puntero a objeto.

```
void*  GetPointer(
   any_class anyobject      // objeto de cualquier clase
   );
```

Parámetros

anyobject

[in] Objeto de cualquier clase.

Valor devuelto

La función devuelve el puntero a objeto.

Nota

Sólo los objetos de clases tienen punteros. Las instancias de estructuras y las variables de tipos simples no tienen punteros. Un objeto de clase que no ha sido creado mediante el operador new(), sino, por ejemplo, ha sido creado automáticamente en el array de objetos, igualmente tiene un puntero. Pero este puntero va a ser del tipo automático POINTER_AUTOMATIC, y no se le puede aplicar el operador delete(). Aparte de eso, este puntero del tipo no tendrá ninguna diferencia de los punteros dinámicos del tipo POINTER_AUTOMATIC.

Debido a que las variables del tipo de estructuras y de tipos simples no tienen punteros, está prohibido aplicarles la función GetPointer(). También está prohibido pasar el puntero como argumento de la función. En todos los casos mencionados el compilador avisará sobre un error.

El intento de llamar a un puntero incorrecto lleva a la terminación crítica del programa, con lo cual es necesario usar la función CheckPointer() antes de utilizar un puntero. Un puntero puede ser incorrecto en las siguientes ocasiones:

- el puntero es igual a NULL;
- si el objeto ha sido eliminado por el operador delete.

Esta función puede ser utilizada para comprobar la validez del puntero. Un valor diferente a cero garantiza que el puntero puede ser utilizado para acceder.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                              Check_GetPointer.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

//+------------------------------------------------------------------+
//|  clase que implementa el elemento de la lista                    |
//+------------------------------------------------------------------+
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))==POINTER_DYNAMIC
                                     "dynamic":"non-dynamic"); }
   void              Initialize(int id,string comm) { m_id=id; m_comment=comm; }
   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; }
  };
//+------------------------------------------------------------------+
//|  la clase más simple de la lista                                 |
//+------------------------------------------------------------------+
class CMyList
  {
   CItem*            m_items;
public:
                     CMyList() { m_items=NULL; }
                    ~CMyList() { Destroy(); }
    bool             InsertToBegin(CItem* item);
    void             Destroy();
  };
//+------------------------------------------------------------------+
//|  inserción del elemento de la lista al principio de todo         |
//+------------------------------------------------------------------+
bool CMyList::InsertToBegin(CItem* item)
  {
   if(CheckPointer(item)==POINTER_INVALID) return(false);
//---
   item.Next(m_items);
   m_items=item;
//---
   return(true);
  }
//+------------------------------------------------------------------+
//|  eliminación de la lista mediante eliminación de sus elementos   |
//+------------------------------------------------------------------+
void CMyList::Destroy()
  {
//--- puntero auxiliar para trabajar en el ciclo
   CItem* item;
//--- pasamos por el ciclo e intentamos eliminar los punteros dinámicos
   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 deleted");
      }
//---
  }
//+------------------------------------------------------------------+
//|  Script program start function                                   |
//+------------------------------------------------------------------+
void OnStart()
  {
   CMyList list;
   CItem   items[10];
   CItem*  item;
//--- creamos y añadimos a la lista un puntero dinámico a objeto
   item=new CItem;
   if(item!=NULL)
     {
      item.Initialize(100,"dynamic");
      item.PrintMe();
      list.InsertToBegin(item);
     }
//--- añadimos los punteros automáticos a la lista
   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);
     }
//--- añadimos otro puntero dinámico a objeto al principio de la lista
   item=new CItem;
   if(item!=NULL)
     {
      item.Initialize(200,"dynamic");
      item.PrintMe();
      list.InsertToBegin(item);
     }
//--- eliminamos los elementos de la lista
   list.Destroy();
//--- todos los elementos de la lista van a ser borrados,
//--- véase la pestaña Experts en el terminal
  }
```

Véase también

Punteros a objetos, Comprobación de puntero a objeto, Operador de eliminación de objeto delete

GetTickCount

La función GetTickCount() devuelve la cantidad de milisegundos pasados desde el momento de arranque del sistema.

```
uint  GetTickCount();
```

Valor devuelto

Valor del tipo uint.

Nota

El contador está limitado por el poder resolutivo del temporizador de sistema. El tiempo se almacena como un entero sin signo, por eso se sobrellena cada 49.7 días con el funcionamiento interrumpido del ordenador.

Ejemplo:

```
#define MAX_SIZE 40
//+------------------------------------------------------------------+
//| un script para medir el tiempo de calculaciones de 40 números de Fibonacci
//+------------------------------------------------------------------+
void OnStart()
  {
//--- recordemos el valor inicial
   uint start=GetTickCount();
//--- variable para obtener el siguiente número de la serie de Fibonacci
   long fib=0;
//--- ciclo en el que se calcula la cantidad especificada de los números de la serie
   for(int i=0;i<MAX_SIZE;i++) fib=TestFibo(i);
//--- obtenemos el tiempo tardado en millisegundos
   uint time=GetTickCount()-start;
//--- mostramos el mensaje en el diario "Asesores Expertos"
   PrintFormat("El cálculo de %d primeros números de Fibonacci ha requerido %d ms",MA
//--- el trabajo del script está finalizado
   return;
  }
//+------------------------------------------------------------------+
//| La función para obtener el número de Fibonacci según su número ordinal      |
//+------------------------------------------------------------------+
long TestFibo(long n)
  {
//--- el primer miembro de la serie de Fibonacci
   if(n<2) return(1);
//--- todos los demás miembros se calculan según la fórmula que viene a continuación
   return(TestFibo(n-2)+TestFibo(n-1));
  }
```

Véase también

Fecha y hora

MessageBox

Crea y visualiza la ventana de mensajes, también lo gestiona. La ventana de mensajes contiene un mensaje y encabezamiento, cualquier combinación de signos predefinidos y botones de dirección.

```
int  MessageBox(
   string  text,              // texto del mensaje
   string  caption=NULL,      // encabezamiento de la ventana
   int     flags=0            // define la combinación de botones en la ventana
   );
```

Parámetros

text

[in] Texto que contiene mensaje para visualizar.

caption=NULL

[in] Texto opcional para visualizar en el encabezamiento de la ventana del mensaje. Si este parámetro está vacío, en el encabezamiento de la ventana se mostrará el nombre del Asesor Experto.

flags=0

[in] Banderas opcionales que determinan la apariencia y comportamiento de la ventana de diálogo. Las banderas pueden ser una combinación de un grupo especial de banderas.

Valor devuelto

Si la función se ejecuta con éxito, el valor devuelto es uno de los valores del código de devolución MessageBox().

Nota

La función no puede ser llamada de los indicadores personalizados, porque los indicadores se ejecutan en un hilo de interfaz y no deben frenarlo.

PeriodSeconds

Devuelve la cantidad de segundos en el período.

```
int  PeriodSeconds(
   ENUM_TIMEFRAMES  period=PERIOD_CURRENT      // período del gráfico
   );
```

Parámetros

period=PERIOD_CURRENT

[in] Valor del período de gráfico de la enumeración ENUM_TIMEFRAMES. Si el parámetro no está especificado, se devuelve el número de segundos del período actual del gráfico en el cual el programa está inicializado.

Valor devuelto

Número de segundos en el período especificado.

Véase también

_Period, Períodos de gráficos, Fecha y hora, Visibilidad de objetos

PlaySound

Reproduce archivo de audio.

```
bool  PlaySound(
   string  filename      // nombre del archivo
   );
```

Parámetros

filename

[in] La ruta del archivo de audio.

Valor devuelto

true - si el archivo de audio ha sido encontrado, en caso contrario devuelve false.

Nota

El archivo tiene que estar ubicado en la carpeta directorio_de_terminal\Sounds o en su subcarpeta. Se reproducen sólo los archivos de audio en el formato WAV.

Véase también

Recursos

Print

Imprime un mensaje en el registro del Asesor Experto. Los parámetros pueden tener cualquier tipo.

```
void  Print(
   argument,      // el primer valor
   ...            // siguientes valores
   );
```

Parámetros

...

[in] Cualquieres valores separados por comas. El número de parámetros no puede superar 64.

Nota

No se puede pasar los arrays a la función Print(). Los arrays deben imprimirse elemento por elemento.

Los datos del tipo double se visualizan con la precisión de hasta 16 dígitos decimales después del punto, además, los datos pueden ser visualizados en el formato tradicional o científico, dependiendo de cuál de ellos va a ser más compacto. Los datos del tipo float se visualizan con 5 dígitos decimales después del punto. Para visualizar los números reales con otra precisión o en un formato explícitamente especificado hay que usar la función PrintFormat().

Los datos del tipo bool se visualizan como cadenas "true" o "false". Las fechas se visualizan como YYYY.MM.DD HH:MI:SS. Para conseguir otro formato de fecha hay que usar la función TimeToString(). Los datos del tipo color se visualizan como cadena R,G,B, o usando el nombre del color si está presente en el juego de colores.

Ejemplo:

```
void OnStart()
  {
//--- imprimimos DBL_MAX utilizando Print(), esto equivale a PrintFormat(%%.16G,DBL_M
   Print("---- como se ve DBL_MAX -----");
   Print("Print(DBL_MAX)=",DBL_MAX);
//--- ahora imprimimos el número DBL_MAX utilizando PrintFormat()
   PrintFormat("PrintFormat(%%.16G,DBL_MAX)=%.16G",DBL_MAX);
//--- Impresión en el diario "Asesores Expertos"
// Print(DBL_MAX)=1.797693134862316e+308
// PrintFormat(%.16G,DBL_MAX)=1.797693134862316E+308

//--- vamos a ver cómo se imprime el tipo float
   float c=(float)M_PI; // hay que convertir explícitamente al tipo de meta
   Print("c=",c, "    Pi=",M_PI, "    (float)M_PI=",(float)M_PI);
// c=3.14159    Pi=3.141592653589793    (float)M_PI=3.14159

//--- mostraremos lo que puede pasar durante las operaciones aritméticas con los tipo
   double a=7,b=200;
   Print("---- antes de las operaciones aritméticas");
   Print("a=",a,"   b=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- vamos a dividir a por b (7/200)
   a=a/b;
//--- ahora como si hubieramos restaurado el valor en la variable b
   b=7.0/a; // se espera que b=7.0/(7.0/200.0)=>7.0/7.0*200.0=200 - pero eso no es as
//--- vamos a imprimir otra vez el valor calculado b
   Print("----- después de las operaciones aritméticas");
   Print("Print(b)=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- impresión en el diario "Asesores Expertos"
// Print(b)=200.0
// Print(DoubleToString(b,16))=199.9999999999999716 (vemos que en realidad b ya no es

//--- vamos a crear un valor muy pequeño epsilon=1E-013
   double epsilon=1e-13;
   Print("---- vamos a crear un número muy pequeño");
   Print("epsilon=",epsilon); // obtenemos   epsilon=1E-013
//--- ahora restamos el épsilon del número b e imprimimos otra vez el valor en el dia
   b=b-epsilon;
//--- imprimimos de dos maneras
   Print("---- después de la sustracción de epsilon de la variable b");
   Print("Print(b)=",b);
   Print("Print(DoubleToString(b,16))=",DoubleToString(b,16));
//--- impresión en el diario "Expertos"
// Print(b)=199.9999999999999  (ahora el valor b después de la sustracción del épsilo
// Print(DoubleToString(b,16))=199.9999999999998578
//    (ahora el valor b después de la sustracción del épsilon no puede ser redondeado
  }
```

Véase también

DoubleToString, StringFormat

PrintFormat

Formatea e imprime juegos de símbolos y valores en el registro histórico del Asesor Experto conforme al formato predefinido.

```
void  PrintFormat(
   string format_string,   // línea de formato
   ...                     // valor de tipos simples
   );
```

Parámetros

format_string

[in] Una línea de formato se compone de símbolos simples y si los argumentos siguen la línea de formato, también contiene la especificación de formato.

...

[in] Cualquieres valores de tipos simples separados por comas. El número total de parámetros no puede superar 64, inclusive la línea de formato.

Valor devuelto

Una cadena.

Nota

La cantidad, orden y tipo de parámetros tiene que corresponder exactamente a la composición de los especificadores, en caso contrario el resultado de la impresión estará indefinida. En vez de la función PrintFormat() se puede usar la función printf().

Si después de la línea de formato siguen los parámetros, esta línea debe contener las especificaciones del formato que denotan el formato de salida de estos parámetros. La especificación del formato siempre se empieza con el signo del porcentaje (%).

Una línea de formato se lee de izquierda a derecha. Al encontrar la primera especificación del formato (si hay), el valor del primer parámetro después de la línea de formato se transforma y se saca conforme a la especificación establecida. La segunda especificación provoca la transformación y salida del segundo parámetro, y así sucesivamente hasta el fin de la línea de formato.

La especificación del formato tiene la siguiente forma:

%[flags][width][.precision][{h | l | ll | I32 | I64}]type

Cada campo de la especificación de formato es un símbolo simple o un número que denota una opción de formato corriente. La especificación de formato más simple contiene sólo el signo de porcentaje (%) y un símbolo que define el tipo de parámetro de salida (por ejemplo %s). Si en la línea de formato hay que mostrar el signo de porcentaje, es necesario usar la especificación de formato %%.

flags

Bandera	Descripción	Comportamiento por

		defecto
- (menos)	Alineación por el lado izquierdo dentro del ancho establecido	Alineación por el lado derecho
+ (más)	Muestra de signos + o - para los tipos con signos	El signo se muestra sólo si el valor es negativo
0 (cero)	Antes de un valor de salida se añaden los ceros dentro del ancho establecido. Si la bandera **0** está especificada con un formato entero (**i**, **u**, **x**, **X**, **o**, **d**) y está determinada la especificación de precisión (por ejemplo, %04.d), entonces 0 se ignora.	Nada se añade
espacio	Antes de un valor de salida se pone un espacio, si el valor es de signo o positivo	Los espacios no se insertan
#	Si se usa junto con el formato **o**, **x** o **X**, entonces antes del valor de salida se añade 0, 0x o 0X respectivamente.	Nada se añade
	Si se usa junto con el formato **e**, **E**, **a** o **A**, el valor siempre se muestra con punto decimal.	El punto decimal se muestra sólo si hay una parte fraccionaria no nula
	Si se usa junto con el formato **g** o **G**, la bandera determina la presencia del punto decimal en el valor de salida e impide el recorte de ceros principales. La bandera # se ignora durante el uso compartido con los formatos **c**, **d**, **i**, **u**, **s**.	El punto decimal se muestra sólo si hay una parte fraccionaria no nula. Los ceros principales se cortan

width

El número decimal no negativo que establece el número mínimo de símbolos de salida del valor formateado. Si el número de símbolos de salida es menos que el ancho especificado, entonces se añade la cantidad correspondiente de espacios a la izquierda o a la derecha dependiendo de la alineación (bandera -). Si hay bandera cero (0), se añade la cantidad correspondiente de ceros antes del valor de salida. Si el número de símbolos de salida es más que el ancho especificado, entonces el valor de salida nunca se corta.

Si el asterisco (*) está especificado como el ancho, el valor del tipo int tiene que estar en la lista de

parámetros pasados en el lugar correspondiente. Este valor va a ser usado para especificar el ancho del valor de salida.

precision

El número decimal no negativo que determina la precisión de salida, es decir, el número de cifras después del punto decimal. A diferencia de la especificación del ancho, la especificación de precisión puede recortar parte del valor fraccionario con redondeo o sin él.

Para diferentes tipos (**type**) de formato la especificación de precisión se aplica de diferentes maneras.

Tipos	descripción	Comportamiento por defecto
a, A	La especificación de precisión fija el número de dígitos después del punto decimal.	Precisión por defecto – 6.
c, C	No se usa.	
d, i, u, o, x, X	Fija el número mínimo de cifras de salida. Si el número de cifras en el parámetro correspondiente es menos de la precisión indicada, el valor de salida se completa con ceros por la izquierda. El valor de salida no se recorta, si el número de cifras de salida es más de la precisión indicada.	Precisión por defecto – 1.
e, E, f	Fija el número de dígitos después del punto decimal. La última cifra se redondea.	Precisión por defecto – 6. Si está especificada la precisión 0 o falta la parte fraccionaria, el punto decimal no se muestra.
g, G	Fija el número máximo de cifras significantes.	Se muestran 6 cifras significantes.
s, S	Fija el número de símbolos de salida de una línea. Si el largo de una línea supera el valor de la precisión, esta línea se recorta en la salida.	Se muestra toda la línea.

h | l | ll | I32 | I64

Especificación de tamaños de datos pasados como parámetros.

Tipo de parámetro	Prefijo utilizado	Especificador compartido del tipo
int	l (L minúscula)	d, i, o, x, or X
uint	l (L minúscula)	o, u, x, or X
long	ll (dos L minúsculas)	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

El especificador del tipo es el único campo obligatorio para la salida formateada.

Símbolo	Tipo	Formato de salida
c	int	Símbolo del tipo short (Unicode)
C	int	Símbolo del tipo char (ANSI)
d	int	Entero decimal con signo
i	int	Entero decimal con signo
o	int	Entero octal sin signo
u	int	Entero decimal sin signo
x	int	Entero hexadecimal sin signo, utilizando "abcdef"
X	int	Entero hexadecimal sin signo, utilizando "ABCDEF"
e	double	Valor real en formato [-] d.dddd **e** [sign]ddd, donde la d es una cifra decimal, dddd - una o más cifras decimales, ddd - número de tres cifras que determina el tamaño de exponente, sign - signo más o menos
E	double	Similar al formato e, salvo que el signo de exponente se

		imprime con mayúscula (**E** en vez de e)
f	double	Valor real en formato [-] dddd.dddd, donde dddd - una o más cifras decimales. El número de dígitos antes del punto decimal depende de la magnitud del valor del número. El número de dígitos después del punto decimal depende la precisión necesaria.
g	double	Valor real mostrado en el formato **f** o **e**, dependiendo de cuál de las salidas va a ser más compacta.
G	double	Valor real mostrado en el formato **f** o **E**, dependiendo de cuál de las salidas va a ser más compacta.
a	double	Valor real en el formato [-] 0xh.hhhh **p**±dd, donde h.hhhh es la mantisa en forma de cifras hexadecimales, usando "abcdef", dd - una o más cifras de la exponente. El número de dígitos después del punto decimal se determina por laespecificación de precisión
A	double	Valor real en el formato [-] 0xh.hhhh **P**±dd, donde h.hhhh es la mantisa en forma de cifras hexadecimales, usando "ABCDEF", dd - una o más cifras de la exponente. El número de dígitos después del punto decimal se determina por la especificación de precisión
s	string	Muestra de línea

En vez de la función PrintFormat() se puede usar la función printf().

Ejemplo:

```
   PrintFormat("Mostramos DBL_MAX en forma científica compacta: %e",DBL_MAX);
   printf("Mostramos DBL_MAX como %%.15e: %.15e",DBL_MAX);
   printf("Mostramos DBL_MAX como %%15e: %15e",DBL_MAX);
   printf("Mostramos DBL_MAX como %%15.10e: %15.10e",DBL_MAX);
   printf("Mostramos DBL_MAX como %%15.10f: %15.10f",DBL_MAX);

   printf("Mostramos 10 como %%f: %f",10);
   printf("Mostramos DBL_MAX como %%d: %d",DBL_MAX);
   printf("Mostramos 10 como %%e: %e",10);
   printf("Mostramos DBL_MAX como %%e: %e",DBL_MAX);
```

Véase también

StringFormat, DoubleToString, Tipos reales (double, float)

ResetLastError

Pone el valor de la variable predefinida _LastError a cero.

```
void  ResetLastError();
```

Valor devuelto

No hay valor devuelto.

Nota

Cabe mencionar que la función GetLastError() no pone a cero la variable _LastError. Habitualmente la función ResetLastError() se invoca antes de llamar a la función después de la cual se comprueba la aparición de un error.

ResourceCreate

Esta función crea un recurso de imagen a base de un conjunto de datos. Hay dos variantes de esta función:

Crear recurso a base de un archivo

```
bool  ResourceCreate(
   const string      resource_name,       // nombre del recurso
   const string      path                 // ruta relativa hacia el archivo
   );
```

Crear archivo a base de un array de píxeles

```
bool  ResourceCreate(
   const string      resource_name,       // nombre del recurso
   const color&      data[],              // conjunto de datos en forma del array
   uint              img_width,           // ancho del recurso de imagen a crear
   uint              img_height,          // alto del recurso de imagen a crear
   uint              data_xoffset,        // desplazamiento horizontal a la derecha d
   uint              data_yoffset,        // desplazamiento vertical hacia abajo de l
   uint              data_width,          // ancho total de la imagen basado en el co
   ENUM_COLOR_FORMAT color_format         // modo de procesar el color
   );
```

Parámetros

resource_name

[in] Nombre del recurso.

data[][]

[in] Un array unidimensional o bidimencional (matriz) de colores para crear una imagen completa.

img_width

[in] Ancho del área rectangular de la imagen en píxeles para colocarse en el recurso en forma de la imagen. No puede ser mayor que el valor de *data_width*.

img_height

[in] Alto del área rectangular de la imagen en píxeles para colocarse en el recurso en forma de la imagen.

data_xoffset

[in] Desplazamiento horizontal a la derecha del área rectangular de la imagen en píxeles.

data_yoffset

[in] Desplazamiento vertical hacia abajo del área rectangular de la imagen en píxeles.

data_width

[in] Se requiere sólo para los arrays unidimensionales, y significa el ancho total de la imagen que se crea a base del conjunto de datos. Si *data_width*=0, entonces se supone que es igual a *img_width*. Para los arrays bidimensionales este parámetro se ignora y se acepta que equivale a la segunda dimensión del array *data[]*.

color_format

[in] Modo de procesar el color desde la enumeración ENUM_COLOR_FORMAT.

Valor devuelto

true – en caso del éxito, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError(). Posibles errores:

- 4015 – ERR_RESOURCE_NAME_DUPLICATED (coincidencia de los nombres del recurso dinámico y estático),
- 4016 – ERR_RESOURCE_NOT_FOUND (recurso no encontrado),
- 4017 – ERR_RESOURCE_UNSUPPOTED_TYPE (tipo del recurso no se soporta),
- 4018 – ERR_RESOURCE_NAME_IS_TOO_LONG (nombre del recurso demasiado largo).

Nota

Si la segunda versión de la función se llama para la creación del mismo recurso con diferentes parámetros del ancho, alto y desplazamiento, el nuevo recurso no se vuelve a crear sino se actualiza el ya existente.

La primera variante de la función permite cargar las imágenes y sonidos desde archivos, mientras que la segunda versión sirve únicamente para la creación dinámica las imágenes.

Las imágenes deben ir en el formato BMP con la profundidad del color de 24 o 32 bits, los sonidos pueden tener sólo el formato WAV. El tamaño del recurso no puede ser más de 16 Mb.

ENUM_COLOR_FORMAT

Identificador	Descripción
COLOR_FORMAT_XRGB_NOALPHA	El componente del canal alfa se ignora
COLOR_FORMAT_ARGB_RAW	Los componentes del color no se procesan por el terminal (deben ser fijados correctamente por el usuario)
COLOR_FORMAT_ARGB_NORMALIZE	Los componentes del color se procesan por el terminal

Véase también

Recursos, ObjectCreate(), ObjectSetString(), OBJPROP_BMPFILE

ResourceSave

Guarda el recurso en el archivo especificado.

```
bool  ResourceSave(
   const string  resource_name      // nombre del recurso
   const string  file_name          // nombre del archivo
   );
```

Parámetros

resource_name

[in] El nombre del recurso debe empezarse con "::".

file_name

[in] Nombre del archivo respecto a MQL5\Files.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

La función siempre sobrescribe el archivo, creando si precisa todas las subcarpetas intermedias en el nombre del archivo, en caso de no haberlas.

Véase también

Recursos, ObjectCreate(), PlaySound(), ObjectSetString(), OBJPROP_BMPFILE

SetUserError

Pone la variable predefinida _LastError en el valor equivalente a ERR_USER_ERROR_FIRST + user_error

```
void  SetUserError(
   ushort user_error,    // número del error
   );
```

Parámetros

user_error

[in] Número del error fijado por el usuario.

Valor devuelto

No hay valor devuelto.

Nota

Después de haber fijado el error con la función SetUserError(user_error), La función GetLastError() devolverá el valor equivalente a ERR_USER_ERROR_FIRST + user_error.

Ejemplo:

```
void OnStart()
  {
//--- fijamos el número del error 65537=(ERR_USER_ERROR_FIRST +1)
   SetUserError(1);
//--- obtendremos el código del último error
   Print("GetLastError = ",GetLastError());
/*
   Resultado
   GetLastError = 65537
*/
  }
```

SendFTP

Envía un archivo a la dirección especificada en la ventana de configuraciones, en la pestaña "Edición".

```
bool  SendFTP(
   string  filename,          // archivo a mandar a través de ftp
   string  ftp_path=NULL      // ruta para la descarga en el servidor-ftp
   );
```

Parámetros

filename

[in] Nombre del archivo a mandar.

ftp_path=NULL

[in] Directorio FTP. Si no está especificado, se utiliza el directorio descrito en las configuraciones.

Valor devuelto

En caso de fallo devuelve false.

Nota

El archivo a mandar tiene que estar ubicado en la carpeta *directorio_de_terminal\MQL5\files* o en sus subcarpetas. El archivo no se envía si la dirección FTP y/o la contraseña de acceso no están especificados en las configuraciones.

SendMail

Envía una carta electrónica a la dirección especificada en la ventana de configuraciones, en la pestaña "Edición".

```
bool  SendMail(
   string  subject,        // asunto
   string  some_text       // texto de la carta
   );
```

Parámetros

subject

[in] Asunto de la carta.

some_text

[in] Cuerpo de la carta.

Valor devuelto

true - si la carta ha sido puesta en cola del envío, en caso contrario devuelve false.

Nota

El envío puede estar prohibido por la configuración, también puede faltar la dirección del correo electrónico. Para obtener la información sobre el error hay que llamar a la función GetLastError().

SendNotification

Esta función envía avisos a los terminales móviles cuyos MetaQuotes ID han sido indicados en la ventana de ajustes en la pestaña "Notificaciones".

```
bool  SendNotification(
   string  text          // texto del mensaje
   );
```

Parámetros

text

[in] Texto del mensaje en la notificación. La longitud del mensaje no puede superar 255 caracteres.

Valor devuelto

Devuelve true si la notificación ha sido enviada con éxito desde el terminal; de lo contrario, devuelve false. Durante la comprobación tras el fallo del envío, GetLastError() puede mostrar uno de los siguientes errores:

- 4515 - ERR_NOTIFICATION_SEND_FAILED,
- 4516 - ERR_NOTIFICATION_WRONG_PARAMETER,
- 4517 - ERR_NOTIFICATION_WRONG_SETTINGS,
- 4518 - ERR_NOTIFICATION_TOO_FREQUENT.

Nota

Para la función SendNotification() existen unas estrictas limitaciones de uso: no más de dos llamadas al segundo y no más de 10 llamadas al minuto. La frecuencia de uso se controla de forma dinámica, y la función puede ser bloqueada si tiene lugar la infracción de estas condiciones.

Sleep

La función suspende la ejecución del Asesor Experto en curso o del script por un intervalo determinado.

```
void  Sleep(
   int  milliseconds      // intervalo
   );
```

Parámetros

milliseconds

[in] Intervalo de retraso en milisegundos.

Valor devuelto

No hay valor devuelto.

Nota

La función Sleep() no puede ser llamada de los indicadores personalizados, porque los indicadores se ejecutan en un hilo de interfaz y no deben frenarlo. La función dispone de la comprobación "built-in" del estado de bandera de interrupción de Asesor Experto cada 0.1 segundos.

TerminalClose

Envía al terminal el comando de finalizar el trabajo.

```
bool  TerminalClose(
   int ret_code      // código de cierre del terminal de cliente
   );
```

Parámetros

ret_code

[in] Código de devolución devuelto por el proceso del terminal de cliente al terminar su trabajo.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

La función TerminalClose() no detiene el terminal inmediatamente, sólo le manda un comando para completar su operatividad.

El código de Asesor Experto que ha llamado a TerminalClose() tiene que realizar todos los preparativos para la finalización inmediata del trabajo (por ejemplo, hay que cerrar todos los archivos abiertos previamente de una forma correcta). Después de la llamada a esta función tiene que seguir el operador return.

El parámetro *ret_code* permite indicar el código de devolución necesario para analizar las causas de finalización programada del funcionamiento del terminal cuando éste se inicia desde la línea de comandos.

Ejemplo:

```
//--- input parameters
input int  tiks_before=500; // número de ticks hasta la finalización
input int  pips_to_go=15;   // distancia en pips
input int  seconds_st=50;   // los segundos que damos al Asesor Experto
//--- 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;
     }
//--- distancia de precio en pips
   distance=(tick.bid-first_bid)/_Point;
//--- mostramos una notificación para monitorear el trabajo del AE
   string comm="Desde el momento de inicio:\r\n\x25CF han pasado segundos: "+
               IntegerToString(tick.time-launch_time)+" ;"+
               "\r\n\x25CF ticks que se ha recibido: "+(string)tick_counter+" ;"+
               "\r\n\x25CF precio ha ido en puntos: "+StringFormat("%G",distance);
   Comment(comm);
//--- sección de comprobación de condiciones para el cierre del terminal
   if(tick_counter>=tiks_before)
      TerminalClose(0);    // salida por el contador de ticks
   if(distance>pips_to_go)
      TerminalClose(1);    // vamos arriba al número de pips igual a pips_to_go
   if(distance<-pips_to_go)
      TerminalClose(-1);   // vamos abajo al número de pips igual a pips_to_go
   if(tick.time-launch_time>seconds_st)
      TerminalClose(100);  // trabajo se termina por expiración de plazo
//---
  }
```

Véase también

Funcionamiento de programas, Errores de ejecución, Razones de deinicialización

TesterStatistics

La función devuelve el valor del indicador estadístico especificado que ha sido calculado a base de los resultados de simulación

```
double  TesterStatistics(
   ENUM_STATISTICS statistic_id      // identificador
   );
```

Parámetros

statistic_id

[in] El identificador del indicador estadístico desde la enumeración ENUM_STATISTICS.

Valor devuelto

El valor del indicador estadístico desde los resultados de simulación.

Nota

La función puede ser llamada dentro de OnTester() o OnDeinit() en el probador de estrategias. En otras ocasiones el resultado no está definido.

TesterWithdrawal

Es una función especial para emular las operaciones de retiro de fondos durante el proceso de evaluación. Puede usarse en algunos sistemas de administración del capital.

```
bool  TesterWithdrawal(
   double money      // importe a retirar
   );
```

Parámetros

money

[in] Cuantía del dinero que hay que retirar de la cuenta (en divisa del depósito).

Valor devuelto

En caso del éxito devuelve true, de lo contrario false.

ZeroMemory

La función reinicializa la variable que le ha sido pasada por referencia.

```
void  ZeroMemory(
   void & variable      // variable reinicializada
   );
```

Parámetros

variable

[in] [out] Variable pasada por referencia a la que hay que reinicializar (inicializar con valores cero).

Valor devuelto

No hay valor devuelto.

Nota

Si el parámetro de la función es una cadena, esta llamada va a ser equivalente a la indicación de NULL como su valor.
Para los tipos simples y sus arrays, así como para las estructuras/clases compuestos de estos tipos, esto es una simple puesta a cero.
Para los objetos que contienen cadenas y arrays dinámicos, ZeroMemory() se llama para cada uno de los elementos.
Para cualquier array que no está protegido por el modificador const, se realiza puesta a cero de todos los elementos.
Para los arrays de objetos complejos, ZeroMemory() se llama para cada uno de los elementos.

La función ZeroMemory() no se aplica a las clases con miembros protegidos o herencia.

Grupo de funciones para trabajar con arrays

Los arrays como máximo pueden ser de cuatro dimensiones. La indexación de cada dimensión se realiza de 0 a *tamaño_de_dimensión-1*. En un caso particular de un array unidimensional de 50 elementos, la referencia hacia el primer elemento se verá como el array[0], hacia el último elemento - array[49].

Función	Acción
ArrayBsearch	Devuelve el índice del primer elemento encontrado en la primera dimensión del array
ArrayCopy	Copia un array al otro
ArrayCompare	Devuelve el resultado de comparación de dos arrays de tipos simples o estructuras personalizadas que no tienen objetos complejos
ArrayFree	Deja libre el búfer de cualquier array dinámico y pone el tamaño de la dimensión cero a 0.
ArrayGetAsSeries	Comprueba la dirección de indexación de un array
ArrayInitialize	Pone todos los elementos de un array numérico en el mismo valor
ArrayFill	Llena un array numérico con valor especificado
ArrayIsSeries	Comprueba si un array es una serie temporal
ArrayIsDynamic	Comprueba si un array es dinámico
ArrayMaximum	Busca un elemento con valor máximo
ArrayMinimum	Busca un elemento con valor mínimo
ArrayRange	Devuelve el número de elementos en la dimensión especificada del array
ArrayResize	Fija nuevo tamaño en la primera dimensión del array
ArraySetAsSeries	Establece la dirección de indexación en el array
ArraySize	Devuelve el número de elementos en el array
ArraySort	Clasificación de arrays numéricos por la primera dimensión

ArrayBsearch

Busca el valor especificado en un array numérico unidimensional clasificado en el orden ascendiente.

Para buscar en un array del tipo double

```
int  ArrayBsearch(
   const double&    array[],   // array para la búsqueda
   double           value      // lo que buscamos
   );
```

Para buscar en un array del tipo float

```
int  ArrayBsearch(
   const float&    array[],   // array para la búsqueda
   float           value      // lo que buscamos
   );
```

Para buscar en un array del tipo long

```
int  ArrayBsearch(
   const long&    array[],   // array para la búsqueda
   long           value      // lo que buscamos
   );
```

Para buscar en un array del tipo int

```
int  ArrayBsearch(
   const int&    array[],   // array para la búsqueda
   int           value      // lo que buscamos
   );
```

Para buscar en un array del tipo short

```
int  ArrayBsearch(
   const short&    array[],   // array para la búsqueda
   short           value      // lo que buscamos
   );
```

Para buscar en un array del tipo char

```
int  ArrayBsearch(
   const char&    array[],   // array para la búsqueda
   char           value      // lo que buscamos
   );
```

Parámetros

array[]

[in] Array numérico para la búsqueda.

value

[in] Valor para la búsqueda.

Valor devuelto

Devuelve el índice del elemento encontrado. Si el valor buscado no ha sido encontrado, la función devuelve el índice del elemento más cercano por el valor.

Nota

La búsqueda binaria procesa sólo los arrays clasificados. Para clasificar un array numérico se utiliza la función ArraySort().

Ejemplo:

```
#property description "Script a base de los datos del indicador RSI muestra en la ven
#property description "la información sobre la frecuencia con la que el mercado se en
#property description "y sobreventa durante el período de tiempo especificado.
//--- mostramos la ventana de los parámetros de entrada durante el arranque del scrip
#property script_show_inputs
//--- parámetros de entrada
input int                InpMAPeriod=14;                  // Período de Media móvil
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;     // Tipo del precio
input double             InpOversoldValue=30.0;           // Nivel de sobreventa
input double             InpOverboughtValue=70.0;         // Nivel de sobrecompra
input datetime           InpDateStart=D'2012.01.01 00:00'; // Fecha de inicio del an
input datetime           InpDateFinish=D'2013.01.01 00:00'; // Fecha de finalización
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   double rsi_buff[]; // array de valores del indicador
   int    size=0;     // tamaño del array
//--- obtenemos el manejador del indicador RSI
   ResetLastError();
   int rsi_handle=iRSI(Symbol(),Period(),InpMAPeriod,InpAppliedPrice);
   if(rsi_handle==INVALID_HANDLE)
     {
      //--- fallo al obtener el manejador del indicador
      PrintFormat("Fallo al obtener el manejador. Código del error = %d",GetLastError
      return;
     }
//--- seguimos en el ciclo hasta que el indicador no termine de calcular todos sus va
   while(BarsCalculated(rsi_handle)==-1)
     {
      //--- salimos si el usuario finaliza el trabajo del script de forma forzada
      if(IsStopped())
         return;
      //--- retardo para que al indicador le de tiempo a calcular sus valores
      Sleep(10);
     }
//--- copiamos los valores del indicador durante un período determinado
   ResetLastError();
   if(CopyBuffer(rsi_handle,0,InpDateStart,InpDateFinish,rsi_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores del indicador. Código del error = %d",
      return;
     }
//--- obtenemos el tamaño del array
   size=ArraySize(rsi_buff);
//--- clasificamos el array
   ArraySort(rsi_buff);
//--- averiguamos el porcentaje de tiempo durante el que el mercado se encuentra en l
   double ovs=(double)ArrayBsearch(rsi_buff,InpOversoldValue)*100/(double)size;
//--- averiguamos el porcentaje de tiempo durante el que el mercado se encuentra en l
   double ovb=(double)(size-ArrayBsearch(rsi_buff,InpOverboughtValue))*100/(double)si
//--- formamos cadenas para visualizar los datos
   string str="De "+TimeToString(InpDateStart,TIME_DATE)+" a "
              +TimeToString(InpDateFinish,TIME_DATE)+" el mercado estaba:";
   string str_ovb="en la zona de sobrecompra "+DoubleToString(ovb,2)+"% de tiempo";
   string str_ovs="en la zona de sobreventa "+DoubleToString(ovs,2)+"% de tiempo";
//--- mostramos los datos en el gráfico
   CreateLabel("top",5,60,str,clrDodgerBlue);
   CreateLabel("overbought",5,35,str_ovb,clrDodgerBlue);
   CreateLabel("oversold",5,10,str_ovs,clrDodgerBlue);
```

```
//--- redibujamos el gráfico
   ChartRedraw(0);
//--- retardo
   Sleep(10000);
  }
//+------------------------------------------------------------------+
//| Mostrar comentario en la esquina inferior izquierda del gráfico
//+------------------------------------------------------------------+
void CreateLabel(const string name,const int x,const int y,
                 const string str,const color clr)
  {
//--- crear etiqueta
   ObjectCreate(0,name,OBJ_LABEL,0,0,0);
//--- anclar etiqueta a la esquina inferior izquierda
   ObjectSetInteger(0,name,OBJPROP_CORNER,CORNER_LEFT_LOWER);
//--- cambiamos la posición del punto de anclaje
   ObjectSetInteger(0,name,OBJPROP_ANCHOR,ANCHOR_LEFT_LOWER);
//--- distancia del punto de anclaje por el eje X
   ObjectSetInteger(0,name,OBJPROP_XDISTANCE,x);
//--- distancia del punto de anclaje por el eje Y
   ObjectSetInteger(0,name,OBJPROP_YDISTANCE,y);
//--- texto de la etiqueta
   ObjectSetString(0,name,OBJPROP_TEXT,str);
//--- color del texto
   ObjectSetInteger(0,name,OBJPROP_COLOR,clr);
//--- tamaño del texto
   ObjectSetInteger(0,name,OBJPROP_FONTSIZE,12);
  }
```

ArrayCopy

Esta función copia un array al otro.

```
int  ArrayCopy(
   void&        dst_array[],          // array de destino
   const void&  src_array[],          // array de origen
   int          dst_start=0,          // índice inicial para copiar desde el array de
   int          src_start=0,          // primer índice del array de destino
   int          count=WHOLE_ARRAY     // número de elementos
   );
```

Parámetros

dst_array[]

[out] Array de destino.

src_array[]

[in] Array de origen.

dst_start=0

[in] Índice inicial para el array de destino. Por defecto, índice inicial - 0.

src_start=0

[in] Índice inicial para el array de origen. Por defecto, índice inicial - 0.

count=WHOLE_ARRAY

[in] Número de elementos que hay que copiar. Por defecto, se copia el array entero (count=WHOLE_ARRAY).

Valor devuelto

Devuelve el número de elementos copiados.

Nota

Si count<0 o count>src_size-src_start, entonces se copia toda la parte restante del array. Los arrays se copian de izquierda a derecha. Para los arrays de serie, la posición de inicio se redefine correctamente tomando en cuenta el copiado de izquierda a derecha. Si un array se copia en si mismo, el resultado es indefinido.

Si los arrays son de tipos diferentes, durante el copiado se intenta la transformación de cada elemento del array de origen en el tipo del array de destino. Los arrays literales se puede copiar sólo a los arrays literales. Los arrays de clases y estructuras que contienen los objetos que requieren la inicialización no se copian. Un array de estructuras puede ser copiado sólo a un array del mismo tipo.

Ejemplo:

```
#property description "El indicador colorea las velas que son"
#property description "máximos y mínimos locales. La distancia del intervalo para bus
#property description "los valores extremos se puede establecer a través del parámetr
//--- ajustes del indicador
```

```
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//---- plot
#property indicator_label1  "Extremums"
#property indicator_type1   DRAW_COLOR_CANDLES
#property indicator_color1  clrLightSteelBlue,clrRed,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- constante predefinida
#define INDICATOR_EMPTY_VALUE 0.0
//--- parámetros de entrada
input int InpNum=4; // Longitud de la mitad del intervalo
//--- búfers indicadores
double ExtOpen[];
double ExtHigh[];
double ExtLow[];
double ExtClose[];
double ExtColor[];
//--- variables globales
int    ExtStart=0; // índice de la primera vela que no es un extremo
int    ExtCount=0; // número de velas no- extremos en este intervalo
//+------------------------------------------------------------------+
//| Coloreado de velas no- extremos                                  |
//+------------------------------------------------------------------+
void FillCandles(const double &open[],const double &high[],
                 const double &low[],const double &close[])
  {
//--- coloreamos velas
   ArrayCopy(ExtOpen,open,ExtStart,ExtStart,ExtCount);
   ArrayCopy(ExtHigh,high,ExtStart,ExtStart,ExtCount);
   ArrayCopy(ExtLow,low,ExtStart,ExtStart,ExtCount);
   ArrayCopy(ExtClose,close,ExtStart,ExtStart,ExtCount);
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtOpen);
   SetIndexBuffer(1,ExtHigh);
   SetIndexBuffer(2,ExtLow);
   SetIndexBuffer(3,ExtClose);
   SetIndexBuffer(4,ExtColor,INDICATOR_COLOR_INDEX);
//--- especificamos el valor que no va a mostrarse
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
//--- especificamos los nombres de los búfers indicadores para mostrar en la ventana
```

```
   PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
//---
   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[])
  {
//--- establecemos la indexación directa en las series temporales
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
//--- variable de inicio para el cálculo de las barras
   int start=prev_calculated;
//--- para las primeras barras InpNum*2 no realizamos el cálculo
   if(start==0)
     {
      start+=InpNum*2;
      ExtStart=0;
      ExtCount=0;
     }
//--- si la barra acaba de formarse, comprobamos el siguiente extremo potencial
   if(rates_total-start==1)
      start--;
//--- índice de la barra cuyo extremo vamos a comprobar
   int ext;
//--- ciclo de cálculo de los valores del indicador
   for(int i=start;i<rates_total-1;i++)
     {
      //--- inicialmente en la barra i sin dibujar
      ExtOpen[i]=0;
      ExtHigh[i]=0;
      ExtLow[i]=0;
      ExtClose[i]=0;
      //--- índice del extremo a chequear
      ext=i-InpNum;
      //--- comprobación del máximo local
```

```
      if(IsMax(high,ext))
        {
         //--- coloreamos la vela extremo
         ExtOpen[ext]=open[ext];
         ExtHigh[ext]=high[ext];
         ExtLow[ext]=low[ext];
         ExtClose[ext]=close[ext];
         ExtColor[ext]=1;
         //--- las demás velas hasta el extremo marcamos con color neutral
         FillCandles(open,high,low,close);
         //--- cambiamos los valores de variables
         ExtStart=ext+1;
         ExtCount=0;
         //--- pasamos a la siguiente iteración
         continue;
        }
      //--- comprobación del mínimo local
      if(IsMin(low,ext))
        {
         //--- coloreamos la vela extremo
         ExtOpen[ext]=open[ext];
         ExtHigh[ext]=high[ext];
         ExtLow[ext]=low[ext];
         ExtClose[ext]=close[ext];
         ExtColor[ext]=2;
         //--- las demás velas hasta el extremo marcamos con color neutral
         FillCandles(open,high,low,close);
         //--- cambiamos los valores de variables
         ExtStart=ext+1;
         ExtCount=0;
         //--- pasamos a la siguiente iteración
         continue;
        }
      //--- incrementamos el número de no-extremos en este intervalo
      ExtCount++;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Comprobamos si el elemento actual del array es un máximo local   |
//+------------------------------------------------------------------+
bool IsMax(const double &price[],const int ind)
  {
//--- variable del inicio del intervalo
   int i=ind-InpNum;
//--- variable del fin del intervalo
   int finish=ind+InpNum+1;
```

```
//--- chequeo para la primera mitad del intervalo
   for(;i<ind;i++)
     {
      if(price[ind]<=price[i])
         return(false);
     }
//--- chequeo para la segunda mitad del intervalo
   for(i=ind+1;i<finish;i++)
     {
      if(price[ind]<=price[i])
         return(false);
     }
//--- es un extremo
   return(true);
  }
//+------------------------------------------------------------------+
//| Comprobamos si el elemento actual del array es un mínimo local   |
//+------------------------------------------------------------------+
bool IsMin(const double &price[],const int ind)
  {
//--- variable del inicio del intervalo
   int i=ind-InpNum;
//--- variable del fin del intervalo
   int finish=ind+InpNum+1;
//--- chequeo para la primera mitad del intervalo
   for(;i<ind;i++)
     {
      if(price[ind]>=price[i])
         return(false);
     }
//--- chequeo para la segunda mitad del intervalo
   for(i=ind+1;i<finish;i++)
     {
      if(price[ind]>=price[i])
         return(false);
     }
//--- es un extremo
   return(true);
  }
```

ArrayCompare

Esta función devuelve el resultado de comparación de dos arrays del mismo tipo. Puede ser utilizada para comparar los arrays de tipos simples o estructuras personalizadas que no tienen objetos complejos, es decir, que no contienen cadenas de caracteres, arrays dinámicos, clases u otras estructuras que incluyen objetos complejos.

```
int  ArrayCompare(
   const void&  array1[],           // el primer array
   const void&  array2[],           // el segundo array
   uint         start1=0,           // offset inicial en el primer array
   uint         start2=0,           // offset inicial en el segundo array
   uint         count=WHOLE_ARRAY   // número de elementos a comparar
   );
```

Parámetros

array1[]

[in] El primer array.

array2[]

[in] El segundo array.

start1=0

[in] Indice inicial del elemento en el primer array a partir del cual se empieza la comparación. Por defecto, el índice inicial es igual a 0.

start2=0

[in] Indice inicial del elemento en el segundo array a partir del cual se empieza la comparación. Por defecto, el índice inicial es igual a 0.

count=WHOLE_ARRAY

[in] Número de elementos a comparar. Por defecto, en la comparación participan todos los elementos de ambos arrays (count=WHOLE_ARRAY).

Valor devuelto

- -1, si array1[] es menor que array2[]
- 0, si array1[] y array2[] son iguales
- 1, si array1[] es mayor a array2[]
- -2, si surge un error debido a la incompatibilidad de tipos de los arrays comparados, o si los valores start1, start2 o count salen de los límites del array.

Nota

La función no va a devolver 0 (los arrays no van a considerarse iguales), si los arrays que se comparan se diferencian en su tamaño y si se indica el valor count=WHOLE_ARRAY para el caso cuando uno de los arrays es el subconjunto exacto del otro. En este caso será devuelto el resultado de comparación de estos arrays: -1, si el tamaño del array1[] es menor que el tamaño del array2[] o 1 en caso contrario.

ArrayFree

Esta función deja libre el búfer de cualquier array dinámico y pone el tamaño de la dimensión cero a 0.

```
void  ArrayFree(
   void&  array[]      // array
   );
```

Parámetros

array[]

[in] Array dinámico.

Valor devuelto

No hay valor devuelto.

Nota

La necesidad de utilizar la función ArrayFree() durante la escritura de los scripts e indicadores puede surgir con poca frecuencia. Es que cuando el trabajo del script se finaliza la memoria utilizada se libera en seguida, el trabajo principal con los arrays en los indicadores personalizados consiste en el acceso a los búferes de indicadores cuyos tamaños se controlan automáticamente por el subsistema ejecutivo del terminal.

Si en el programa es necesario administrar la memoria manualmente en unas condiciones dinámicas complicadas, la función ArrayFree()permite al usuario liberar la memoria ocupada por un array dinámico ya innecesario de forma explícita e inmediata.

Ejemplo:

```
#include <Controls\Dialog.mqh>
#include <Controls\Button.mqh>
#include <Controls\Label.mqh>
#include <Controls\ComboBox.mqh>
//--- constantes predefinidas
#define X_START 0
#define Y_START 0
#define X_SIZE 280
#define Y_SIZE 300
//+------------------------------------------------------------------+
//| Clase de diálogo para trabajar con la memoria                    |
//+------------------------------------------------------------------+
class CMemoryControl : public CAppDialog
  {
private:
   //--- tamaño del array
   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[];
   //--- etiquetas
   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;
   //--- botones
   CButton           m_button_add;
   CButton           m_button_free;
   //--- cajas combinadas
   CComboBox         m_combo_box_step;
   CComboBox         m_combo_box_type;
   //--- valor actual del tipo del array de la caja combinada
   int               m_combo_box_type_value;

public:
                     CMemoryControl(void);
                    ~CMemoryControl(void);
   //--- método de creación del objeto de la clase
   virtual bool      Create(const long chart,const string name,const int subwin,const
   //--- manejador de eventos del gráfico
   virtual bool      OnEvent(const int id,const long &lparam,const double &dparam,con

protected:
   //--- crear etiqueta
   bool              CreateLabel(CLabel &lbl,const string name,const int x,const int
   //--- crear elementos de control
   bool              CreateButton(CButton &button,const string name,const int x,const
   bool              CreateComboBoxStep(void);
   bool              CreateComboBoxType(void);
   //--- manejadores de eventos
   void              OnClickButtonAdd(void);
   void              OnClickButtonFree(void);
   void              OnChangeComboBoxType(void);
   //--- métodos de trabajo con array actual
   void              CurrentArrayFree(void);
   bool              CurrentArrayAdd(void);
  };
//+------------------------------------------------------------------+
//| Liberación de memoria del array actual                           |
//+------------------------------------------------------------------+
void CMemoryControl::CurrentArrayFree(void)
```

```
  {
//--- reiniciar el tamaño del array
   m_arr_size=0;
//--- liberación del 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);
  }
//+------------------------------------------------------------------+
//| Intento de agregar memoria para array actual                     |
//+------------------------------------------------------------------+
bool CMemoryControl::CurrentArrayAdd(void)
  {
//--- salir si el tamaño de la memoria utilizada supera el tamaño de la memoria físic
   if(TerminalInfoInteger(TERMINAL_MEMORY_PHYSICAL)/TerminalInfoInteger(TERMINAL_MEMO
      return(false);
//--- intento de adjudicación de la memoria de acuerdo con el tipo actual
   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);
//--- memoria adjudicada
   return(true);
  }
//+------------------------------------------------------------------+
//| Manejo de eventos                                                |
//+------------------------------------------------------------------+
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)
  {
  }
//+------------------------------------------------------------------+
//| Método de creación del objeto de la clase                        |
//+------------------------------------------------------------------+
bool CMemoryControl::Create(const long chart,const string name,const int subwin,
                            const int x1,const int y1,const int x2,const int y2)
  {
//--- creación del objeto de la clase basa
   if(!CAppDialog::Create(chart,name,subwin,x1,y1,x2,y2))
      return(false);
//--- preparación de las cadenas para las etiquetas
   string str_physical="Memory physical = "+(string)TerminalInfoInteger(TERMINAL_MEMO
   string str_total="Memory total = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_TOT
   string str_available="Memory available = "+(string)TerminalInfoInteger(TERMINAL_ME
   string str_used="Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMORY_USED)
//--- crear etiquetas
   if(!CreateLabel(m_lbl_memory_physical,"physical_label",X_START+10,Y_START+5,str_ph
      return(false);
   if(!CreateLabel(m_lbl_memory_total,"total_label",X_START+10,Y_START+30,str_total,1
      return(false);
   if(!CreateLabel(m_lbl_memory_available,"available_label",X_START+10,Y_START+55,str
      return(false);
   if(!CreateLabel(m_lbl_memory_used,"used_label",X_START+10,Y_START+80,str_used,12,c
      return(false);
   if(!CreateLabel(m_lbl_array_type,"type_label",X_START+10,Y_START+105,"Array type =
      return(false);
   if(!CreateLabel(m_lbl_array_size,"size_label",X_START+10,Y_START+130,"Array size =
      return(false);
   if(!CreateLabel(m_lbl_error,"error_label",X_START+10,Y_START+155,"",12,clrRed))
      return(false);
   if(!CreateLabel(m_lbl_change_type,"change_type_label",X_START+10,Y_START+185,"Chan
      return(false);
   if(!CreateLabel(m_lbl_add_size,"add_size_label",X_START+10,Y_START+210,"Add to arr
      return(false);
//--- crear elementos de control
   if(!CreateButton(m_button_add,"add_button",X_START+15,Y_START+245,"Add",12,clrBlue
      return(false);
   if(!CreateButton(m_button_free,"free_button",X_START+75,Y_START+245,"Free",12,clrB
      return(false);
   if(!CreateComboBoxType())
```

```
      return(false);
   if(!CreateComboBoxStep())
      return(false);
//--- inicialización de variable
   m_arr_size=0;
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| Crear el botón                                                   |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateButton(CButton &button,const string name,const int x,
                                  const int y,const string str,const int font_size,
                                  const int clr)
  {
//--- crear el botón
   if(!button.Create(m_chart_id,name,m_subwin,x,y,x+50,y+20))
      return(false);
//--- texto
   if(!button.Text(str))
      return(false);
//--- tamaño de la fuente
   if(!button.FontSize(font_size))
      return(false);
//--- color de la etiqueta
   if(!button.Color(clr))
      return(false);
//--- agregamos el botón a los elementos de control
   if(!Add(button))
      return(false);
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| Crear la caja combinada para el tamaño del array                 |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateComboBoxStep(void)
  {
//--- crear la caja combinada
   if(!m_combo_box_step.Create(m_chart_id,"step_combobox",m_subwin,X_START+100,Y_STAR
      return(false);
//--- agregar elementos a la caja combinada
   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);
//--- establecer el elemento actual de la caja combinada
   if(!m_combo_box_step.SelectByValue(1000000))
      return(false);
//--- agregamos la caja combinada a los elementos de control
   if(!Add(m_combo_box_step))
      return(false);
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| Crear la caja combinada para el tipo del array                   |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateComboBoxType(void)
  {
//--- crear la caja combinada
   if(!m_combo_box_type.Create(m_chart_id,"type_combobox",m_subwin,X_START+100,Y_STAR
      return(false);
//--- agregar elementos a la caja combinada
   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);
//--- establecer el elemento actual de la caja combinada
   if(!m_combo_box_type.SelectByValue(3))
      return(false);
//--- memorizamos el elemento actual de la caja combinada
   m_combo_box_type_value=3;
//--- agregamos la caja combinada a los elementos de control
   if(!Add(m_combo_box_type))
      return(false);
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| Crear etiqueta                                                   |
//+------------------------------------------------------------------+
bool CMemoryControl::CreateLabel(CLabel &lbl,const string name,const int x,
                                 const int y,const string str,const int font_size,
                                 const int clr)
  {
```

```
//--- crear etiqueta
   if(!lbl.Create(m_chart_id,name,m_subwin,x,y,0,0))
      return(false);
//--- texto
   if(!lbl.Text(str))
      return(false);
//--- tamaño de la fuente
   if(!lbl.FontSize(font_size))
      return(false);
//--- color
   if(!lbl.Color(clr))
      return(false);
//--- agregamos etiqueta a los elementos de control
   if(!Add(lbl))
      return(false);
//--- succeed
   return(true);
  }
//+------------------------------------------------------------------+
//| Manejador del evento de pulsación del botón "Add"                |
//+------------------------------------------------------------------+
void CMemoryControl::OnClickButtonAdd(void)
  {
//--- aumentamos el tamaño del array
   m_arr_size+=(int)m_combo_box_step.Value();
//--- intentamos adjudicar la memoria para el array actual
   if(CurrentArrayAdd())
     {
      //--- memoria adjudicada, mostramos el estado actual sobre la pantalla
      m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(T
      m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_ME
      m_lbl_array_size.Text("Array size = "+IntegerToString(m_arr_size));
      m_lbl_error.Text("");
     }
   else
     {
      //--- no se ha podido adjudicar la memoria, mostramos mensaje sobre el error
      m_lbl_error.Text("Array is too large, error!");
      //--- devolvemos el tamaño anterior del array
      m_arr_size-=(int)m_combo_box_step.Value();
     }
  }
//+------------------------------------------------------------------+
//| Manejador del evento de pulsación del botón "Free"               |
//+------------------------------------------------------------------+
void CMemoryControl::OnClickButtonFree(void)
  {
//--- liberamos la memoria del array actual
```

```
   CurrentArrayFree();
//--- mostramos el estado actual sobre la pantalla
   m_lbl_memory_available.Text("Memory available = "+(string)TerminalInfoInteger(TERM
   m_lbl_memory_used.Text("Memory used = "+(string)TerminalInfoInteger(TERMINAL_MEMOR
   m_lbl_array_size.Text("Array size = 0");
   m_lbl_error.Text("");
  }
//+------------------------------------------------------------------+
//| Manejador del evento de la modificación de caja combinada
//+------------------------------------------------------------------+
void CMemoryControl::OnChangeComboBoxType(void)
  {
//--- prueba, si se ha cambiado el tipo del array
   if(m_combo_box_type.Value()!=m_combo_box_type_value)
     {
      //--- liberamos la memoria del array actual
      OnClickButtonFree();
      //--- trabajamos con otro tipo del array
      m_combo_box_type_value=(int)m_combo_box_type.Value();
      //--- mostramos nuevo tipo del array en la pantalla
      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");
     }
  }
//--- objeto de la clase CMemoryControl
CMemoryControl ExtDialog;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- crear diálogo
   if(!ExtDialog.Create(0,"MemoryControl",0,X_START,Y_START,X_SIZE,Y_SIZE))
      return(INIT_FAILED);
//--- iniciar
   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

Esta función comprueba la dirección de indexación de un array.

```
bool  ArrayGetAsSeries(
   const void&  array[]    // array a comprobar
   );
```

Parámetros

array

[in] Array comprobado.

Valor devuelto

Devuelve true, si el array especificado tiene puesta la bandera AS_SERIES, es decir, el acceso al array se realiza al revés, como en una serie temporal. Una serie temporal se diferencia de un array usual en que la indexación de los elementos de la serie temporal se realiza del fin del array al inicio (de los datos más recientes a los más antiguos).

Nota

Para comprobar si un array pertenece a una serie temporal hay que usar la función ArrayIsSeries(). Los arrays de los datos de precio que han sido pasados como los parámetros de entrada a la función OnCalculate(), no han de tener obligatoriamente la dirección de indexación como las series temporales. La función ArraySetAsSeries() puede establecer la dirección de indexación necesaria.

Ejemplo:

```
#property description "El indicador calcula los valores absolutos de diferencia entre
#property description "Open y Close o High y Low, y los visualiza en una subventana i
#property description "en forma de un histograma."
//--- ajustes del indicador
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_style1  STYLE_SOLID
#property indicator_width1  3
//--- parámetros de entrada
input bool InpAsSeries=true; // Dirección de indexación en el búfer de indicadores
input bool InpPrices=true;   // Precios para el cálculo (true - Open,Close; false - H
//--- búfer de indicadores
double ExtBuffer[];
//+------------------------------------------------------------------+
//| Cálculo de valores del indicador                                 |
//+------------------------------------------------------------------+
void CandleSizeOnBuffer(const int rates_total,const int prev_calculated,
                        const double &first[],const double &second[],double &buffer[]
  {
//--- variable inicial para el cálculo de barras
   int start=prev_calculated;
//--- si los valores del indicador ya han sido calculados en el tick anterior, entonc
   if(prev_calculated>0)
      start--;
//--- definimos la dirección de la indexación en los arrays
   bool as_series_first=ArrayGetAsSeries(first);
   bool as_series_second=ArrayGetAsSeries(second);
   bool as_series_buffer=ArrayGetAsSeries(buffer);
//--- si hace falta, sustituimos la dirección de la indexación con la recta
   if(as_series_first)
      ArraySetAsSeries(first,false);
   if(as_series_second)
      ArraySetAsSeries(second,false);
   if(as_series_buffer)
      ArraySetAsSeries(buffer,false);
//--- calculamos los valores del indicador
   for(int i=start;i<rates_total;i++)
      buffer[i]=MathAbs(first[i]-second[i]);
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- enlace de los búferes de indicadores
   SetIndexBuffer(0,ExtBuffer);
//--- fijamos la dirección de la indexación en el búfer de indicadores
   ArraySetAsSeries(ExtBuffer,InpAsSeries);
//--- comprobamos para qué tipo de precios se calcula el indicador
   if(InpPrices)
     {
      //--- precios Open y Close
      PlotIndexSetString(0,PLOT_LABEL,"BodySize");
      //--- determinamos el color del indicador
      PlotIndexSetInteger(0,PLOT_LINE_COLOR,clrOrange);
     }
   else
     {
```

```
      //--- precios High y Low
      PlotIndexSetString(0,PLOT_LABEL,"ShadowSize");
      //--- determinamos el color del indicador
      PlotIndexSetInteger(0,PLOT_LINE_COLOR,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[])
  {
//--- cálculo del indicador en función del valor de la bandera
   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);
  }
```

Véase también

Acceso a las series temporales, ArraySetAsSeries

ArrayInitialize

Esta función inicializa un array numérico usando un valor especificado.

```
void  ArrayInitialize(
   double  array[],     // array inicializado
   double  value        // valor que va a ser puesto
   );
```

Parámetros

array[]

[out] Array numérico que hay que inicializar.

value

[in] Nuevo valor que hay que asignar a todos los elementos del array.

Valor devuelto

No hay valor devuelto.

Nota

La función ArrayResize() permite establecer el tamaño de un array con una cierta reserva para su futura expansión sin redistribución física de la memoria. Esto se implementa para mejorar el rendimiento, puesto que las operaciones de redistribución de la memoria son bastante lentas.

La inicialización del array usando la expresión ArrayInitialize(array, init_val) no significa la inicialización con el mismo valor de los elementos de la reserva adjudicada para este array. Con las futuras expansiones del tamaño del array *array* mediante la función ArrayResize() dentro de los márgenes de la reserva actual, al final del array se añadirán los elementos cuyos valores no estarán definidos, y en mayoría de las ocasiones no van a ser iguales a *init_val*.

Ejemplo:

```
void OnStart()
  {
//--- array dinámico
   double array[];
//--- vamos a establecer el tamaño del array para 100 elementos y reservar el búfer p
   ArrayResize(array,100,10);
//--- inicializamos los elementos del array con el valor EMPTY_VALUE=DBL_MAX
   ArrayInitialize(array,EMPTY_VALUE);
   Print("Valores de los últimos 10 elementos del array después de la inicialización"
   for(int i=90;i<100;i++) printf("array[%d] = %G",i,array[i]);
//--- aumentamos el tamaño del array por 5 elementos
   ArrayResize(array,105);
   Print("Valores de los últimos 10 elementos del array después de ArrayResize(array,
//--- los valores de 5 últimos elementos han sido recibidos desde el buffer de reserv
   for(int i=95;i<105;i++) printf("array[%d] = %G",i,array[i]);
  }
```

ArrayFill

Llena un array numérico con valor especificado.

```
void  ArrayFill(
   void&  array[],       // array
   uint   start,         // índice del elemento inicial
   uint   count,         // número de elementos
   void   value          // valor con el que se llena el array
   );
```

Parámetros

array[]

[out] Array del tipo simple (char, uchar, short, ushort, int, uint, long, ulong, bool, color, datetime, float, double).

start

[in] Índice del elemento inicial (a partir del cual empezamos a llenar). En este caso, la bandera de series establecida se ignora.

count

[in] Número de elementos a rellenar.

value

[in] Valor con el que se llena el array.

Valor devuelto

No hay valor devuelto.

Nota

Cuando se llama a la función ArrayFill(), siempre se sobreentiende la dirección habitual de la indexación (de izquierda a derecha). Es decir, el cambio del orden de acceso a los elementos del array mediante la función ArraySetAsSeries() no se toma en consideración.

Un array multidimensional se representa como unidimensional cuando se procesa por la función ArrayFill(). Por ejemplo, el array array[2][4] se procesa como array[8]. Por eso a la hora de trabajar con este array se puede especificar el índice del elemento inicial igual a 5. De esta manera, la llamada a ArrayFill(array, 5, 2, 3.14) para el array array[2][4] va a llenar los elementos del array array[1][1] y array[1][2] con el valor 3.14.

Ejemplo:

```
void OnStart()
  {
//--- declaramos un array dinámico
   int a[];
//--- establecemos el tamaño
   ArrayResize(a,10);
//--- llenamos 5 elementos iniciales con el valor 123
   ArrayFill(a,0,5,123);
//--- llenamos 5 elementos (a partir del quinto) con el valor 456
   ArrayFill(a,5,5,456);
```

```
//--- mostramos los valores de todos los elementos
   for(int i=0;i<ArraySize(a);i++) printf("a[%d] = %d",i,a[i]);
  }
```

ArrayIsDynamic

La función comprueba si el array es dinámico.

```
bool  ArrayIsDynamic(
   const void&  array[]    // array comprobado
   );
```

Parámetros

array[]

[in] Array que se comprueba.

Valor devuelto

Devuelve true, si el array indicado es dinámico, de lo contrario devuelve false.

Ejemplo:

```
#property description "Este indicador no calcula valores, sino intenta aplicar una ve
#property description "la llamada de la función ArrayFree() a tres arrays: dinámico, 
#property description "búfer de indicadores. Los resultados se muestran en el diario 
//--- ajustes del indicador
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- variables globales
double ExtDynamic[];   // array dinámico
double ExtStatic[100]; // array estático
bool   ExtFlag=true;   // bandera
double ExtBuff[];      // búfer de indicadores
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- adjudicamos la memoria para el 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 int begin,
                const double &price[])
  {
//--- realizamos el análisis singular
   if(ExtFlag)
     {
      //--- tratamos de liberar la memoria para los arrays
      //--- 1. Array dinámico
      Print("+============================+");
      Print("1. Cheque del array dinámico:");
      Print("Tamaño antes de liberar memoria = ",ArraySize(ExtDynamic));
      Print("Es un array dinámico = ",ArrayIsDynamic(ExtDynamic) ? "Sí" : "No");
      //--- tratamos de liberar la memoria del array
      ArrayFree(ExtDynamic);
      Print("Tamaño después de liberar memoria = ",ArraySize(ExtDynamic));
      //--- 2. Array estático
      Print("2. Chequeo del array estático:");
      Print("Tamaño antes de liberar memoria = ",ArraySize(ExtStatic));
      Print("Es un array estático = ",ArrayIsDynamic(ExtStatic) ? "Sí" : "No");
      //--- tratamos de liberar la memoria del array
      ArrayFree(ExtStatic);
      Print("Tamaño después de liberar memoria = ",ArraySize(ExtStatic));
      //--- 3. Búfer de indicadores
      Print("3. Chequeo del búfer de indicadores:");
      Print("Tamaño antes de liberar memoria = ",ArraySize(ExtBuff));
      Print("Es un array dinámico = ",ArrayIsDynamic(ExtBuff) ? "Sí" : "No");
      //--- tratamos de liberar la memoria del array
      ArrayFree(ExtBuff);
      Print("Tamaño después de liberar memoria = ",ArraySize(ExtBuff));
      //--- cambiamos el valor de la bandera
      ExtFlag=false;
```

```
      }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Véase también

Acceso a las series temporales e indicadores

ArrayIsSeries

Esta función comprueba si el array es una serie temporal.

```
bool  ArrayIsSeries(
   const void&  array[]    // array comprobado
   );
```

Parámetros

array[]

[in] Array que se comprueba.

Valor devuelto

Devuelve true, si el array comprobado es un array-serie temporal, de lo contrario devuelve false. Los arrays pasados como un parámetro a la función OnCalculate(), tienen que ser comprobados para el orden de acceso a los elementos del array con la función ArrayGetAsSeries().

Ejemplo:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   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);
  }
```

Véase también

Acceso a las series temporales e indicadores

ArrayMaximum

Esta función busca el elemento máximo en un array numérico unidimensional.

```
int  ArrayMaximum(
   const void&   array[],            // array para la búsqueda
   int           start=0,            // a partir de qué índice empezamos a buscar
   int           count=WHOLE_ARRAY   // número de elementos a comprobar
   );
```

Parámetros

array[]

[in] Array numérico donde se realiza la búsqueda.

start=0

[in] Índice de partida para la búsqueda.

count=WHOLE_ARRAY

[in] Número de elementos para la búsqueda. Por defecto, buscamos en todo el array (count=WHOLE_ARRAY).

Valor devuelto

La función devuelve el índice del elemento encontrado teniendo en cuenta la serie del array. En caso del fallo la función devuelve -1.

Ejemplo:

```
#property description "El indicador muestra las velas del mayor período en el período
//--- ajustes del indicador
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots   8
//---- plot 1
#property indicator_label1  "BearBody"
#property indicator_color1  clrSeaGreen,clrSeaGreen
//---- plot 2
#property indicator_label2  "BearBodyEnd"
#property indicator_color2  clrSeaGreen,clrSeaGreen
//---- plot 3
#property indicator_label3  "BearShadow"
#property indicator_color3  clrSalmon,clrSalmon
//---- plot 4
#property indicator_label4  "BearShadowEnd"
#property indicator_color4  clrSalmon,clrSalmon
//---- plot 5
#property indicator_label5  "BullBody"
#property indicator_color5  clrOlive,clrOlive
//---- plot 6
#property indicator_label6  "BullBodyEnd"
```

```
#property indicator_color6  clrOlive,clrOlive
//---- plot 7
#property indicator_label7  "BullShadow"
#property indicator_color7  clrSkyBlue,clrSkyBlue
//---- plot 8
#property indicator_label8  "BullShadowEnd"
#property indicator_color8  clrSkyBlue,clrSkyBlue
//--- constante predefinida
#define INDICATOR_EMPTY_VALUE 0.0
//--- parámetros de entrada
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4;              // Período para el cálculo de
input datetime        InpDateStart=D'2013.01.01 00:00'; // Fecha de inicio del anális
//--- búferes de indicadores para las velas bajistas
double   ExtBearBodyFirst[];
double   ExtBearBodySecond[];
double   ExtBearBodyEndFirst[];
double   ExtBearBodyEndSecond[];
double   ExtBearShadowFirst[];
double   ExtBearShadowSecond[];
double   ExtBearShadowEndFirst[];
double   ExtBearShadowEndSecond[];
//--- búferes de indicadores para las velas alcistas
double   ExtBullBodyFirst[];
double   ExtBullBodySecond[];
double   ExtBullBodyEndFirst[];
double   ExtBullBodyEndSecond[];
double   ExtBullShadowFirst[];
double   ExtBullShadowSecond[];
double   ExtBullShadowEndFirst[];
double   ExtBullShadowEndSecond[];
//--- variables globales
datetime ExtTimeBuff[];      // búfer de tiempo del período mayor
int      ExtSize=0;          // tamaño del búfer de tiempo
int      ExtCount=0;         // índice dentro del búfer de tiempo
int      ExtStartPos=0;      // posición inicial para el cálculo del indicador
bool     ExtStartFlag=true;  // bandera auxiliar para recibir la posición inicial
datetime ExtCurrentTime[1];  // última hora de formación de la barra desde el período
datetime ExtLastTime;        // última hora desde el período mayor para el que se ha
bool     ExtBearFlag=true;   // bandera para determinar el orden de escritura de dato
bool     ExtBullFlag=true;   // bandera para determinar el orden de escritura de dato
int      ExtIndexMax=0;      // índice del elemento máximo del array
int      ExtIndexMin=0;      // índice del elemento mínimo del array
int      ExtDirectionFlag=0; // dirección de movimiento del precio para la vela actua
//--- espacio entre el precio de apertura y cierre de la vela para una correcta repre
const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+------------------------------------------------------------------+
//| Coloreado de la parte básica de la vela                          |
//+------------------------------------------------------------------+
```

```
void FillCandleMain(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    int &index_max,int &index_min)
  {
//--- buscamos los índices del elemento máximo y del mínimo en los arrays
   index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // el máximo en High
   index_min=ArrayMinimum(low,ExtStartPos,last-start+1);  // el mínimo en Low
//--- determinamos el número de barras desde el período actual que vamos a colear
   int count=fill_index-start+1;
//--- si el precio de cierre de la primera barra supera el precio de cierre de la últ
   if(open[start]>close[last])
     {
      //--- si hasta este momento la vela era bajista, entonces limpiamos los valores
      if(ExtDirectionFlag!=-1)
         ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullSha
      //--- vela bajista
      ExtDirectionFlag=-1;
      //--- formamos la vela
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearSha
                     close[last],high[index_max],low[index_min],start,count,ExtBearFl
      //--- salida de la función
      return;
     }
//--- si el precio de cierre de la primera barra es inferior al precio de cierre de l
   if(open[start]<close[last])
     {
      //--- si hasta este momento la vela era alcistas, entonces limpiamos los valore
      if(ExtDirectionFlag!=1)
         ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearSha
      //--- vela alcista
      ExtDirectionFlag=1;
      //--- formamos la vela
      FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullSha
                     open[start],high[index_max],low[index_min],start,count,ExtBullFl
      //--- salida de la función
      return;
     }
//--- si se encuentra en esta parte de la función, el precio de apertura de la primer
//--- precio de cierre de la última barra; vamos a considerar esta vela bajista
//--- si hasta este momento la vela era bajista, entonces limpiamos los valores de lo
   if(ExtDirectionFlag!=-1)
      ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadow
//--- vela bajista
   ExtDirectionFlag=-1;
//--- si los precios de cierre y de apertura son iguales, utilizamos el desplazamient
   if(high[index_max]!=low[index_min])
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearSha
```

```
                    open[start]-ExtEmptyBodySize,high[index_max],low[index_min],star
   else
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearSha
                    open[start],open[start]-ExtEmptyBodySize,high[index_max],
                    high[index_max]-ExtEmptyBodySize,start,count,ExtBearFlag);
  }
//+------------------------------------------------------------------+
//| Coloreado de la punta de la vela                                 |
//+------------------------------------------------------------------+
void FillCandleEnd(const double &open[],const double &close[],
                   const double &high[],const double &low[],
                   const int start,const int last,const int fill_index,
                   const int index_max,const int index_min)
  {
//--- no dibujamos en caso de sólo una barra
   if(last-start==0)
      return;
//--- si el precio de cierre de la primera barra supera el precio de cierre de la últ
   if(open[start]>close[last])
     {
      //--- formamos la punta de la vela
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ex
                    open[start],close[last],high[index_max],low[index_min],fill_index
      //--- salida de la función
      return;
     }
//--- si el precio de cierre de la primera barra es inferior al precio de cierre de l
   if(open[start]<close[last])
     {
      //--- formamos la punta de la vela
      FormCandleEnd(ExtBullBodyEndFirst,ExtBullBodyEndSecond,ExtBullShadowEndFirst,Ex
                    close[last],open[start],high[index_max],low[index_min],fill_index
      //--- salida de la función
      return;
     }
//--- si se encuentra en esta parte de la función, el precio de apertura de la primer
//--- precio de cierre de la última barra; vamos a considerar esta vela bajista
//--- formamos la punta de la vela
   if(high[index_max]!=low[index_min])
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ex
                    open[start]-ExtEmptyBodySize,high[index_max],low[index_min],fill_
   else
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ex
                    open[start]-ExtEmptyBodySize,high[index_max],high[index_max]-ExtE
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
```

```
int OnInit()
  {
//--- chequeo del período de tiempo del indicador
   if(!CheckPeriod((int)Period(),(int)InpPeriod))
      return(INIT_PARAMETERS_INCORRECT);
//--- visualización de los precios en el primer plano
   ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- enlace de los búferes de indicadores
   SetIndexBuffer(0,ExtBearBodyFirst);
   SetIndexBuffer(1,ExtBearBodySecond);
   SetIndexBuffer(2,ExtBearBodyEndFirst);
   SetIndexBuffer(3,ExtBearBodyEndSecond);
   SetIndexBuffer(4,ExtBearShadowFirst);
   SetIndexBuffer(5,ExtBearShadowSecond);
   SetIndexBuffer(6,ExtBearShadowEndFirst);
   SetIndexBuffer(7,ExtBearShadowEndSecond);
   SetIndexBuffer(8,ExtBullBodyFirst);
   SetIndexBuffer(9,ExtBullBodySecond);
   SetIndexBuffer(10,ExtBullBodyEndFirst);
   SetIndexBuffer(11,ExtBullBodyEndSecond);
   SetIndexBuffer(12,ExtBullShadowFirst);
   SetIndexBuffer(13,ExtBullShadowSecond);
   SetIndexBuffer(14,ExtBullShadowEndFirst);
   SetIndexBuffer(15,ExtBullShadowEndSecond);
//--- fijamos algunos valores de propiedades para construir el indicador
   for(int i=0;i<8;i++)
     {
      PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // tipo de construcción grá
      PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // estilo de la línea
      PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // grosor de la línea
     }
//---
   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[])
  {
```

```
//--- si todavía no hay barras calculadas,
   if(prev_calculated==0)
     {
      //--- obtenemos la hora de aparición de las barras desde el período mayor
      if(!GetTimeData())
         return(0);
     }
//--- establecemos la dirección directa para la indexación
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(close,false);
//--- variable de inicio para el cálculo de las barras
   int start=prev_calculated;
//--- si la barra se está formando, volvemos a calcular el valor del indicador sobre
   if(start!=0 && start==rates_total)
      start--;
//--- ciclo de cálculo de los valores del indicador
   for(int i=start;i<rates_total;i++)
     {
      //--- llenamos los elementos "i" de los búferes de indicadores con valores vací
      FillIndicatorBuffers(i);
      //--- hacemos el cálculo para las barras a partir de la fecha InpDateStart
      if(time[i]>=InpDateStart)
        {
         //--- definimos por primera vez la posición a partir de la cual empezamos a
         if(ExtStartFlag)
           {
            //--- recordamos el número de la barra inicial
            ExtStartPos=i;
            //--- determinamos la primera fecha desde el período mayor que supera tim
            while(time[i]>=ExtTimeBuff[ExtCount])
               if(ExtCount<ExtSize-1)
                  ExtCount++;
            //--- cambiamos el valor de la bandera para no volver a entrar en este bl
            ExtStartFlag=false;
           }
         //--- comprobamos si hay más elementos en el array
         if(ExtCount<ExtSize)
           {
            //--- esperamos hasta que el valor de tiempo del período actual alcance e
            if(time[i]>=ExtTimeBuff[ExtCount])
              {
               //--- dibujamos la parte básica de la vela (sin colorear entre la últi
               FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,Ext
               //--- coloreamos la punta de la vela (área entre la última barra y la
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtI
```

```
               //--- movemos la posición inicial para dibujar la siguiente barra
               ExtStartPos=i;
               //--- aumentamos el contador del array
               ExtCount++;
              }
            else
               continue;
           }
         else
           {
            //--- anulamos los valores del error
            ResetLastError();
            //--- obtenemos la última fecha del período mayor
            if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
              {
               Print("Error del copiado de datos, código = ",GetLastError());
               return(0);
              }
            //--- si nueva fecha es mayor, terminamos la formación de la vela
            if(ExtCurrentTime[0]>ExtLastTime)
              {
               //--- limpiamos el área entre la última barra y la penúltima en los bú
               ClearEndOfBodyMain(i-1);
               //--- coloreamos esta área utilizando los búferes de indicadores auxil
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtI
               //--- movemos la posición inicial para dibujar la siguiente barra
               ExtStartPos=i;
               //--- reseteamos la bandera de dirección del precio
               ExtDirectionFlag=0;
               //--- recordamos la última fecha nueva
               ExtLastTime=ExtCurrentTime[0];
              }
            else
              {
               //--- formamos la vela
               FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtInde
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Prueba de la corrección del período del indicador introducido    |
//+------------------------------------------------------------------+
bool CheckPeriod(int current_period,int high_period)
  {
```

```
//--- el período del indicador tiene que ser más grande que el período de tiempo (tim
   if(current_period>=high_period)
     {
      Print("¡Error! ¡El valor del período del indicador tiene que superar al valor d
      return(false);
     }
//--- si el período del indicador es una semana o un mes, entonces el período es corr
   if(high_period>32768)
      return(true);
//--- convertimos los valores de los períodos a los minutos
   if(high_period>30)
      high_period=(high_period-16384)*60;
   if(current_period>30)
      current_period=(current_period-16384)*60;
//--- el período del indicador debe de ser múltiple del período de tiempo en el que s
   if(high_period%current_period!=0)
     {
      Print("¡Error! ¡El valor del período del indicador tiene que múltiple del valor
      return(false);
     }
//--- el período del indicador tiene que superar al período de tiempo (time frame) en
   if(high_period/current_period<3)
     {
      Print("¡Error! ¡El valor del período del indicador tiene que superar al valor d
      return(false);
     }
período del indicador es correcto para el período de tiempo actual
   return(true);
  }
//+------------------------------------------------------------------+
//| Recepción de datos de tiempo desde el período de tiempo mayor    |
//+------------------------------------------------------------------+
bool GetTimeData(void)
  {
//--- resetear el valor del error
   ResetLastError();
//--- copiamos todos los datos para la hora actual
   if(CopyTime(Symbol(),InpPeriod,InpDateStart,TimeCurrent(),ExtTimeBuff)==-1)
     {
      //--- obtenemos el código del error
      int code=GetLastError();
      //--- imprimimos el texto del error
      PrintFormat("¡Error al copiar datos! %s",code==4401
                  ? "¡El historial se sigue cargando!"
                  : "Código = "+IntegerToString(code));
      //--- devolvemos false para volver a intentar cargar datos
      return(false);
     }
```

```
//--- obtenemos el tamaño del array
   ExtSize=ArraySize(ExtTimeBuff);
//--- ponemos el índice del ciclo para el array igual a cero
   ExtCount=0;
//--- ponemos la posición de la vela actual en este período de tiempo igual a cero
   ExtStartPos=0;
   ExtStartFlag=true;
//--- recordamos el último valor de tiempo desde el período de tiempo mayor
   ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función forma la parte básica de la vela. Dependiendo del valor de|
//| la bandera, la función determina qué datos y en qué arrays deben   |
//| escribirse para una visualización correcta.                        |
//+------------------------------------------------------------------+
void FormCandleMain(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int start,const int count,const bool flag)
  {
//--- comprobamos el valor de la bandera
   if(flag)
     {
      //--- formamos el cuerpo de la vela
      FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
      //--- formamos la sombra de la vela
      FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
     }
   else
     {
      //--- formamos el cuerpo de la vela
      FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
      //--- formamos la sombra de la vela
      FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
     }
  }
//+------------------------------------------------------------------+
//| La función forma la punta de la vela. Dependiendo del valor de la bandera,  |
//| la función determina qué datos y en qué arrays deben          |
//| escribirse para una visualización correcta.                        |
//+------------------------------------------------------------------+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                   double &shadow_fst[],double &shadow_snd[],
                   const double fst_value,const double snd_value,
                   const double fst_extremum,const double snd_extremum,
```

```
                    const int end,bool &flag)
  {
//--- comprobamos el valor de la bandera
   if(flag)
     {
      //--- formamos el fin del cuerpo de la vela
      FormEnd(body_fst,body_snd,fst_value,snd_value,end);
      //--- formamos el fin de la sombra de la vela
      FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
      //--- cambiamos el valor de la bandera al opuesto
      flag=false;
     }
   else
     {
      //--- formamos el fin del cuerpo de la vela
      FormEnd(body_fst,body_snd,snd_value,fst_value,end);
      //--- formamos el fin de la sombra de la vela
      FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
      //--- cambiamos el valor de la bandera al opuesto
      flag=true;
     }
  }
//+------------------------------------------------------------------+
//| Limpiamos la punta de la vela (área entre la última barra y la penúltima     |
//| barra)                                                           |
//+------------------------------------------------------------------+
void ClearEndOfBodyMain(const int ind)
  {
   ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSec
   ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSec
  }
//+------------------------------------------------------------------+
//| Limpieza de la vela                                              |
//+------------------------------------------------------------------+
void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
  {
//--- prueba
   if(count!=0)
     {
      //--- llenamos los búferes indicadores con valores vacíos
      ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
     }
  }
//+------------------------------------------------------------------+
```

```
//| Formación de la parte básica de la vela                          |
//+------------------------------------------------------------------+
void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)
  {
//--- prueba
   if(count!=0)
     {
      //--- llenamos los búferes indicadores con valores
      ArrayFill(fst,start,count,fst_value);
      ArrayFill(snd,start,count,snd_value);
     }
  }
//+------------------------------------------------------------------+
//| Formación de la punta de la vela                                 |
//+------------------------------------------------------------------+
void FormEnd(double &fst[],double &snd[],const double fst_value,
             const double snd_value,const int last)
  {
//--- llenamos los búferes indicadores con valores
   ArrayFill(fst,last-1,2,fst_value);
   ArrayFill(snd,last-1,2,snd_value);
  }
//+------------------------------------------------------------------+
//| Llenar los elementos "i" de los búferes de indicadores con valores vacíos|
//+------------------------------------------------------------------+
void FillIndicatorBuffers(const int i)
  {
//--- establecemos un valor vacío en la cédula de búferes de indicadores
   ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
  }
```

ArrayMinimum

Esta función busca un elemento mínimo en un array numérico unidimensional.

```
int  ArrayMinimum(
   const void&   array[],            // array para la búsqueda
   int           start=0,            // a partir de qué índice empezamos a buscar
   int           count=WHOLE_ARRAY   // número de elementos a comprobar
   );
```

Parámetros

array[]

[in] Array numérico donde se realiza la búsqueda.

start=0

[in] Índice de partida para la búsqueda.

count=WHOLE_ARRAY

[in] Número de elementos para la búsqueda. Por defecto, buscamos en todo el array (count=WHOLE_ARRAY).

Valor devuelto

La función devuelve el índice del elemento encontrado teniendo en cuenta la serie del array. En caso del fallo la función devuelve -1.

Ejemplo:

```
#property description "El indicador muestra las velas del mayor período en el período
//--- ajustes del indicador
#property indicator_chart_window
#property indicator_buffers 16
#property indicator_plots   8
//---- plot 1
#property indicator_label1  "BearBody"
#property indicator_color1  clrSeaGreen,clrSeaGreen
//---- plot 2
#property indicator_label2  "BearBodyEnd"
#property indicator_color2  clrSeaGreen,clrSeaGreen
//---- plot 3
#property indicator_label3  "BearShadow"
#property indicator_color3  clrSalmon,clrSalmon
//---- plot 4
#property indicator_label4  "BearShadowEnd"
#property indicator_color4  clrSalmon,clrSalmon
//---- plot 5
#property indicator_label5  "BullBody"
#property indicator_color5  clrOlive,clrOlive
//---- plot 6
#property indicator_label6  "BullBodyEnd"
```

```
#property indicator_color6  clrOlive,clrOlive
//---- plot 7
#property indicator_label7  "BullShadow"
#property indicator_color7  clrSkyBlue,clrSkyBlue
//---- plot 8
#property indicator_label8  "BullShadowEnd"
#property indicator_color8  clrSkyBlue,clrSkyBlue
//--- constante predefinida
#define INDICATOR_EMPTY_VALUE 0.0
//--- parámetros de entrada
input ENUM_TIMEFRAMES InpPeriod=PERIOD_H4;              // Período para el cálculo de
input datetime        InpDateStart=D'2013.01.01 00:00'; // Fecha de inicio del anális
//--- búferes de indicadores para las velas bajistas
double   ExtBearBodyFirst[];
double   ExtBearBodySecond[];
double   ExtBearBodyEndFirst[];
double   ExtBearBodyEndSecond[];
double   ExtBearShadowFirst[];
double   ExtBearShadowSecond[];
double   ExtBearShadowEndFirst[];
double   ExtBearShadowEndSecond[];
//--- búferes de indicadores para las velas alcistas
double   ExtBullBodyFirst[];
double   ExtBullBodySecond[];
double   ExtBullBodyEndFirst[];
double   ExtBullBodyEndSecond[];
double   ExtBullShadowFirst[];
double   ExtBullShadowSecond[];
double   ExtBullShadowEndFirst[];
double   ExtBullShadowEndSecond[];
//--- variables globales
datetime ExtTimeBuff[];      // búfer de tiempo del período mayor
int      ExtSize=0;          // tamaño del búfer de tiempo
int      ExtCount=0;         // índice dentro del búfer de tiempo
int      ExtStartPos=0;      // posición inicial para el cálculo del indicador
bool     ExtStartFlag=true;  // bandera auxiliar para recibir la posición inicial
datetime ExtCurrentTime[1];  // última hora de formación de la barra desde el período
datetime ExtLastTime;        // última hora desde el período mayor para el que se ha
bool     ExtBearFlag=true;   // bandera para determinar el orden de escritura de dato
bool     ExtBullFlag=true;   // bandera para determinar el orden de escritura de dato
int      ExtIndexMax=0;      // índice del elemento máximo del array
int      ExtIndexMin=0;      // índice del elemento mínimo del array
int      ExtDirectionFlag=0; // dirección de movimiento del precio para la vela actua
//--- espacio entre el precio de apertura y cierre de la vela para una correcta repre
const double ExtEmptyBodySize=0.2*SymbolInfoDouble(Symbol(),SYMBOL_POINT);
//+------------------------------------------------------------------+
//| Coloreado de la parte básica de la vela                          |
//+------------------------------------------------------------------+
```

```
void FillCandleMain(const double &open[],const double &close[],
                    const double &high[],const double &low[],
                    const int start,const int last,const int fill_index,
                    int &index_max,int &index_min)
  {
//--- buscamos los índices del elemento máximo y del mínimo en los arrays
   index_max=ArrayMaximum(high,ExtStartPos,last-start+1); // el máximo en High
   index_min=ArrayMinimum(low,ExtStartPos,last-start+1);  // el mínimo en Low
//--- determinamos el número de barras desde el período actual que vamos a colear
   int count=fill_index-start+1;
//--- si el precio de cierre de la primera barra supera el precio de cierre de la últ
   if(open[start]>close[last])
     {
      //--- si hasta este momento la vela era bajista, entonces limpiamos los valores
      if(ExtDirectionFlag!=-1)
         ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullSha
      //--- vela bajista
      ExtDirectionFlag=-1;
      //--- formamos la vela
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearSha
                     close[last],high[index_max],low[index_min],start,count,ExtBearFl
      //--- salida de la función
      return;
     }
//--- si el precio de cierre de la primera barra es inferior al precio de cierre de l
   if(open[start]<close[last])
     {
      //--- si hasta este momento la vela era alcistas, entonces limpiamos los valore
      if(ExtDirectionFlag!=1)
         ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearSha
      //--- vela alcista
      ExtDirectionFlag=1;
      //--- formamos la vela
      FormCandleMain(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullSha
                     open[start],high[index_max],low[index_min],start,count,ExtBullFl
      //--- salida de la función
      return;
     }
//--- si se encuentra en esta parte de la función, el precio de apertura de la primer
//--- precio de cierre de la última barra; vamos a considerar esta vela bajista
//--- si hasta este momento la vela era bajista, entonces limpiamos los valores de lo
   if(ExtDirectionFlag!=-1)
      ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadow
//--- vela bajista
   ExtDirectionFlag=-1;
//--- si los precios de cierre y de apertura son iguales, utilizamos el desplazamient
   if(high[index_max]!=low[index_min])
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearSha
```

```
                     open[start]-ExtEmptyBodySize,high[index_max],low[index_min],star
   else
      FormCandleMain(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearSha
                     open[start],open[start]-ExtEmptyBodySize,high[index_max],
                     high[index_max]-ExtEmptyBodySize,start,count,ExtBearFlag);
  }
//+------------------------------------------------------------------+
//| Coloreado de la punta de la vela                                 |
//+------------------------------------------------------------------+
void FillCandleEnd(const double &open[],const double &close[],
                   const double &high[],const double &low[],
                   const int start,const int last,const int fill_index,
                   const int index_max,const int index_min)
  {
//--- no dibujamos en caso de sólo una barra
   if(last-start==0)
      return;
//--- si el precio de cierre de la primera barra supera el precio de cierre de la últ
   if(open[start]>close[last])
     {
      //--- formamos la punta de la vela
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ex
                    open[start],close[last],high[index_max],low[index_min],fill_index
      //--- salida de la función
      return;
     }
//--- si el precio de cierre de la primera barra es inferior al precio de cierre de l
   if(open[start]<close[last])
     {
      //--- formamos la punta de la vela
      FormCandleEnd(ExtBullBodyEndFirst,ExtBullBodyEndSecond,ExtBullShadowEndFirst,Ex
                    close[last],open[start],high[index_max],low[index_min],fill_index
      //--- salida de la función
      return;
     }
//--- si se encuentra en esta parte de la función, el precio de apertura de la primer
//--- precio de cierre de la última barra; vamos a considerar esta vela bajista
//--- formamos la punta de la vela
   if(high[index_max]!=low[index_min])
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ex
                    open[start]-ExtEmptyBodySize,high[index_max],low[index_min],fill_
   else
      FormCandleEnd(ExtBearBodyEndFirst,ExtBearBodyEndSecond,ExtBearShadowEndFirst,Ex
                    open[start]-ExtEmptyBodySize,high[index_max],high[index_max]-ExtE
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
```

```
int OnInit()
  {
//--- chequeo del período de tiempo del indicador
   if(!CheckPeriod((int)Period(),(int)InpPeriod))
      return(INIT_PARAMETERS_INCORRECT);
//--- visualización de los precios en el primer plano
   ChartSetInteger(0,CHART_FOREGROUND,0,1);
//--- enlace de los búferes de indicadores
   SetIndexBuffer(0,ExtBearBodyFirst);
   SetIndexBuffer(1,ExtBearBodySecond);
   SetIndexBuffer(2,ExtBearBodyEndFirst);
   SetIndexBuffer(3,ExtBearBodyEndSecond);
   SetIndexBuffer(4,ExtBearShadowFirst);
   SetIndexBuffer(5,ExtBearShadowSecond);
   SetIndexBuffer(6,ExtBearShadowEndFirst);
   SetIndexBuffer(7,ExtBearShadowEndSecond);
   SetIndexBuffer(8,ExtBullBodyFirst);
   SetIndexBuffer(9,ExtBullBodySecond);
   SetIndexBuffer(10,ExtBullBodyEndFirst);
   SetIndexBuffer(11,ExtBullBodyEndSecond);
   SetIndexBuffer(12,ExtBullShadowFirst);
   SetIndexBuffer(13,ExtBullShadowSecond);
   SetIndexBuffer(14,ExtBullShadowEndFirst);
   SetIndexBuffer(15,ExtBullShadowEndSecond);
//--- fijamos algunos valores de propiedades para construir el indicador
   for(int i=0;i<8;i++)
     {
      PlotIndexSetInteger(i,PLOT_DRAW_TYPE,DRAW_FILLING); // tipo de construcción grá
      PlotIndexSetInteger(i,PLOT_LINE_STYLE,STYLE_SOLID); // estilo de la línea
      PlotIndexSetInteger(i,PLOT_LINE_WIDTH,1);           // grosor de la línea
     }
//---
   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[])
  {
```

```
//--- si todavía no hay barras calculadas,
   if(prev_calculated==0)
     {
      //--- obtenemos la hora de aparición de las barras desde el período mayor
      if(!GetTimeData())
         return(0);
     }
//--- establecemos la dirección directa para la indexación
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(close,false);
//--- variable de inicio para el cálculo de las barras
   int start=prev_calculated;
//--- si la barra se está formando, volvemos a calcular el valor del indicador sobre
   if(start!=0 && start==rates_total)
      start--;
//--- ciclo de cálculo de los valores del indicador
   for(int i=start;i<rates_total;i++)
     {
      //--- llenamos los elementos "i" de los búferes de indicadores con valores vací
      FillIndicatorBuffers(i);
      //--- hacemos el cálculo para las barras a partir de la fecha InpDateStart
      if(time[i]>=InpDateStart)
        {
         //--- definimos por primera vez la posición a partir de la cual empezamos a
         if(ExtStartFlag)
           {
            //--- recordamos el número de la barra inicial
            ExtStartPos=i;
            //--- determinamos la primera fecha desde el período mayor que supera tim
            while(time[i]>=ExtTimeBuff[ExtCount])
               if(ExtCount<ExtSize-1)
                  ExtCount++;
            //--- cambiamos el valor de la bandera para no volver a entrar en este bl
            ExtStartFlag=false;
           }
         //--- comprobamos si hay más elementos en el array
         if(ExtCount<ExtSize)
           {
            //--- esperamos hasta que el valor de tiempo del período actual alcance e
            if(time[i]>=ExtTimeBuff[ExtCount])
              {
               //--- dibujamos la parte básica de la vela (sin colorear entre la últi
               FillCandleMain(open,close,high,low,ExtStartPos,i-1,i-2,ExtIndexMax,Ext
               //--- coloreamos la punta de la vela (área entre la última barra y la
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtI
```

```
               //--- movemos la posición inicial para dibujar la siguiente barra
               ExtStartPos=i;
               //--- aumentamos el contador del array
               ExtCount++;
              }
            else
               continue;
           }
         else
           {
            //--- anulamos los valores del error
            ResetLastError();
            //--- obtenemos la última fecha del período mayor
            if(CopyTime(Symbol(),InpPeriod,0,1,ExtCurrentTime)==-1)
              {
               Print("Error del copiado de datos, código = ",GetLastError());
               return(0);
              }
            //--- si nueva fecha es mayor, terminamos la formación de la vela
            if(ExtCurrentTime[0]>ExtLastTime)
              {
               //--- limpiamos el área entre la última barra y la penúltima en los bú
               ClearEndOfBodyMain(i-1);
               //--- coloreamos esta área utilizando los búferes de indicadores auxil
               FillCandleEnd(open,close,high,low,ExtStartPos,i-1,i-1,ExtIndexMax,ExtI
               //--- movemos la posición inicial para dibujar la siguiente barra
               ExtStartPos=i;
               //--- reseteamos la bandera de dirección del precio
               ExtDirectionFlag=0;
               //--- recordamos la última fecha nueva
               ExtLastTime=ExtCurrentTime[0];
              }
            else
              {
               //--- formamos la vela
               FillCandleMain(open,close,high,low,ExtStartPos,i,i,ExtIndexMax,ExtInde
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Prueba de la corrección del período del indicador introducido    |
//+------------------------------------------------------------------+
bool CheckPeriod(int current_period,int high_period)
  {
```

```
//--- el período del indicador tiene que ser más grande que el período de tiempo (tim
   if(current_period>=high_period)
     {
      Print("¡Error! ¡El valor del período del indicador tiene que superar al valor d
      return(false);
     }
//--- si el período del indicador es una semana o un mes, entonces el período es corr
   if(high_period>32768)
      return(true);
//--- convertimos los valores de los períodos a los minutos
   if(high_period>30)
      high_period=(high_period-16384)*60;
   if(current_period>30)
      current_period=(current_period-16384)*60;
//--- el período del indicador debe de ser múltiple del período de tiempo en el que s
   if(high_period%current_period!=0)
     {
      Print("¡Error! ¡El valor del período del indicador tiene que múltiple del valor
      return(false);
     }
//--- el período del indicador tiene que superar al período de tiempo (time frame) en
   if(high_period/current_period<3)
     {
      Print("¡Error! ¡El valor del período del indicador tiene que superar al valor d
      return(false);
     }
período del indicador es correcto para el período de tiempo actual
   return(true);
  }
//+------------------------------------------------------------------+
//| Recepción de datos de tiempo desde el período de tiempo mayor    |
//+------------------------------------------------------------------+
bool GetTimeData(void)
  {
//--- resetear el valor del error
   ResetLastError();
//--- copiamos todos los datos para la hora actual
   if(CopyTime(Symbol(),InpPeriod,InpDateStart,TimeCurrent(),ExtTimeBuff)==-1)
     {
      //--- obtenemos el código del error
      int code=GetLastError();
      //--- imprimimos el texto del error
      PrintFormat("¡Error al copiar datos! %s",code==4401
                  ? "¡El historial se sigue cargando!"
                  : "Código = "+IntegerToString(code));
      //--- devolvemos false para volver a intentar cargar datos
      return(false);
     }
```

```
//--- obtenemos el tamaño del array
   ExtSize=ArraySize(ExtTimeBuff);
//--- ponemos el índice del ciclo para el array igual a cero
   ExtCount=0;
//--- ponemos la posición de la vela actual en este período de tiempo igual a cero
   ExtStartPos=0;
   ExtStartFlag=true;
//--- recordamos el último valor de tiempo desde el período de tiempo mayor
   ExtLastTime=ExtTimeBuff[ExtSize-1];
//--- ejecución con éxito
   return(true);
  }
//+------------------------------------------------------------------+
//| La función forma la parte básica de la vela. Dependiendo del valor de|
//| la bandera, la función determina qué datos y en qué arrays deben   |
//| escribirse para una visualización correcta.                        |
//+------------------------------------------------------------------+
void FormCandleMain(double &body_fst[],double &body_snd[],
                    double &shadow_fst[],double &shadow_snd[],
                    const double fst_value,const double snd_value,
                    const double fst_extremum,const double snd_extremum,
                    const int start,const int count,const bool flag)
  {
//--- comprobamos el valor de la bandera
   if(flag)
     {
      //--- formamos el cuerpo de la vela
      FormMain(body_fst,body_snd,fst_value,snd_value,start,count);
      //--- formamos la sombra de la vela
      FormMain(shadow_fst,shadow_snd,fst_extremum,snd_extremum,start,count);
     }
   else
     {
      //--- formamos el cuerpo de la vela
      FormMain(body_fst,body_snd,snd_value,fst_value,start,count);
      //--- formamos la sombra de la vela
      FormMain(shadow_fst,shadow_snd,snd_extremum,fst_extremum,start,count);
     }
  }
//+------------------------------------------------------------------+
//| La función forma la punta de la vela. Dependiendo del valor de la bandera,  |
//| la función determina qué datos y en qué arrays deben          |
//| escribirse para una visualización correcta.                        |
//+------------------------------------------------------------------+
void FormCandleEnd(double &body_fst[],double &body_snd[],
                   double &shadow_fst[],double &shadow_snd[],
                   const double fst_value,const double snd_value,
                   const double fst_extremum,const double snd_extremum,
```

```
                    const int end,bool &flag)
  {
//--- comprobamos el valor de la bandera
   if(flag)
     {
      //--- formamos el fin del cuerpo de la vela
      FormEnd(body_fst,body_snd,fst_value,snd_value,end);
      //--- formamos el fin de la sombra de la vela
      FormEnd(shadow_fst,shadow_snd,fst_extremum,snd_extremum,end);
      //--- cambiamos el valor de la bandera al opuesto
      flag=false;
     }
   else
     {
      //--- formamos el fin del cuerpo de la vela
      FormEnd(body_fst,body_snd,snd_value,fst_value,end);
      //--- formamos el fin de la sombra de la vela
      FormEnd(shadow_fst,shadow_snd,snd_extremum,fst_extremum,end);
      //--- cambiamos el valor de la bandera al opuesto
      flag=true;
     }
  }
//+------------------------------------------------------------------+
//| Limpiamos la punta de la vela (área entre la última barra y la penúltima     |
//| barra)                                                           |
//+------------------------------------------------------------------+
void ClearEndOfBodyMain(const int ind)
  {
   ClearCandle(ExtBearBodyFirst,ExtBearBodySecond,ExtBearShadowFirst,ExtBearShadowSec
   ClearCandle(ExtBullBodyFirst,ExtBullBodySecond,ExtBullShadowFirst,ExtBullShadowSec
  }
//+------------------------------------------------------------------+
//| Limpieza de la vela                                              |
//+------------------------------------------------------------------+
void ClearCandle(double &body_fst[],double &body_snd[],double &shadow_fst[],
                 double &shadow_snd[],const int start,const int count)
  {
//--- prueba
   if(count!=0)
     {
      //--- llenamos los búferes indicadores con valores vacíos
      ArrayFill(body_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(body_snd,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_fst,start,count,INDICATOR_EMPTY_VALUE);
      ArrayFill(shadow_snd,start,count,INDICATOR_EMPTY_VALUE);
     }
  }
//+------------------------------------------------------------------+
```

```
//| Formación de la parte básica de la vela                          |
//+------------------------------------------------------------------+
void FormMain(double &fst[],double &snd[],const double fst_value,
              const double snd_value,const int start,const int count)
  {
//--- prueba
   if(count!=0)
     {
      //--- llenamos los búferes indicadores con valores
      ArrayFill(fst,start,count,fst_value);
      ArrayFill(snd,start,count,snd_value);
     }
  }
//+------------------------------------------------------------------+
//| Formación de la punta de la vela                                 |
//+------------------------------------------------------------------+
void FormEnd(double &fst[],double &snd[],const double fst_value,
             const double snd_value,const int last)
  {
//--- llenamos los búferes indicadores con valores
   ArrayFill(fst,last-1,2,fst_value);
   ArrayFill(snd,last-1,2,snd_value);
  }
//+------------------------------------------------------------------+
//| Llenar los elementos "i" de los búferes de indicadores con valores vacíos|
//+------------------------------------------------------------------+
void FillIndicatorBuffers(const int i)
  {
//--- establecemos un valor vacío en la cédula de búferes de indicadores
   ExtBearBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBearShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodySecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullBodyEndSecond[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndFirst[i]=INDICATOR_EMPTY_VALUE;
   ExtBullShadowEndSecond[i]=INDICATOR_EMPTY_VALUE;
  }
```

ArrayRange

Esta función devuelve el número de elementos en la dimensión especificada del array.

```
int  ArrayRange(
   const void&   array[],      // array a comprobar
   int           rank_index    // número de dimensión
   );
```

Parámetros

array[]

[in] Array a comprobar.

rank_index

[in] Índice de dimensión.

Valor devuelto

Número de elementos en la dimensión especificada del array.

Nota

Puesto que los índices se empiezan desde cero, el número de dimensiones del array es a uno más grande que el índice de la última dimensión.

Ejemplo:

```
void OnStart()
  {
//--- creamos un array de 4 dimensiones
   double array[][5][2][4];
//--- fijamos el tamaño de la dimensión-0
   ArrayResize(array,10,10);
//--- imprimimos dimensiones
   int temp;
   for(int i=0;i<4;i++)
     {
      //--- obtenemos el tamaño de dimensión i
      temp=ArrayRange(array,i);
      //--- imprimimos
      PrintFormat("dim = %d, range = %d",i,temp);
     }
//--- Resultado
// dim = 0, range = 10
// dim = 1, range = 5
// dim = 2, range = 2
// dim = 3, range = 4
  }
```

ArrayResize

Esta función establece nuevo tamaño en la primera dimensión del array.

```
int  ArrayResize(
   void&  array[],              // array pasado por referencia
   int    new_size,             // nuevo tamaño del array
   int    reserve_size=0        // valor de reserva del tamaño (sobrante)
   );
```

Parámetros

array[]

[out] Array para el cambio de tamaño.

new_size

[in] Nuevo tamaño para la primera dimensión.

reserve_size=0

[in] Tamaño para la reserva adicional.

Valor devuelto

Si se ejecuta con éxito, la función devuelve la cantidad de todos los elementos contenidos en el array después del cambio de tamaño; de lo contrario devuelve -1 y el array no cambia sus dimensiones.

Nota

Esta función puede aplicarse sólo a los arrays dinámicos. Además, hay que tener en cuenta que no se puede cambiar el tamaño de los arrays dinámicos que han sido asignados como búfers de indicadores por la función SetIndexBuffer(). Todas las operaciones relacionadas con el cambio del tamaño de los búfers de indicadores se realizan por el subsistema ejecutivo del terminal.

En caso de la distribución frecuente de la memoria se recomienda utilizar el tercer parámetro que establece una reserva para disminuir la cantidad de distribución física de la memoria. Las siguientes llamadas a la función ArrayResize no llevan a la redistribución física de la memoria, simplemente se cambia el tamaño de la primera dimensión del array dentro de los límites de la memoria reservada. Hay que recordar que el tercer parámetro va a utilizarse sólo cuando va a tener lugar la distribución física de la memoria, por ejemplo:

```
ArrayResize(arr,1000,1000);
for(int i=1;i<3000;i++)
   ArrayResize(arr,i,1000);
```

En este caso se producirá 2 redistribuciones de la memoria: una vez antes de la entrada en el ciclo de 2000 elementos, en este caso la dimensionalidad del array se establecerá en 1000, y la segunda - cuando i es igual a 2000. Si omitimos el tercer parámetro, habrá 2000 redistribuciones físicas de la memoria, y esto ralentizará la ejecución del programa.

Ejemplo:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- contadores
   ulong start=GetTickCount();
   ulong now;
   int   count=0;
//--- array para demostración de la versión rápida
   double arr[];
   ArrayResize(arr,100000,100000);
//--- comprobamos con qué rapidez funciona la versión con la reserva de memoria
   Print("--- Test Fast: ArrayResize(arr,100000,100000)");
   for(int i=1;i<=300000;i++)
     {
      //--- ¡fijamos el nuevo tamaño del array indicando la reserva de 100000 element
      ArrayResize(arr,i,100000);
      //--- al alcanzar un número redondo, mostramos el tamaño del array y el tiempo
      if(ArraySize(arr)%100000==0)
        {
         now=GetTickCount();
         count++;
         PrintFormat("%d. ArraySize(arr)=%d Time=%d ms",count,ArraySize(arr),(now-sta
         start=now;
        }
     }
//--- ahora mostramos qué lento trabaja la versión sin la reserva de la memoria
   double slow[];
   ArrayResize(slow,100000,100000);
//---
   count=0;
   start=GetTickCount();
   Print("---- Test Slow: ArrayResize(slow,100000)");
//---
   for(int i=1;i<=300000;i++)
     {
      //--- fijamos el nuevo tamaño del array, pero ya sin la reserva adicional
      ArrayResize(slow,i);
      //--- al alcanzar un número redondo, mostramos el tamaño del array y el tiempo
      if(ArraySize(slow)%100000==0)
        {
         now=GetTickCount();
         count++;
         PrintFormat("%d. ArraySize(slow)=%d Time=%d ms",count,ArraySize(slow),(now-s
         start=now;
        }
     }
  }
//--- Un resultado aproximado de la ejecución del script
/*
   Test_ArrayResize (EURUSD,H1)   --- Test Fast: ArrayResize(arr,100000,100000)
   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
*/
```

Véase también

ArrayInitialize

ArraySetAsSeries

Esta función pone la bandera AS_SERIES al objeto del array dinámico especificado, la indexación de los elementos del array va a efectuarse como en las series temporales.

```
bool  ArraySetAsSeries(
   const void&  array[],     // array por referencia
   bool         flag         // true significa el orden inverso de indexación
   );
```

Parámetros

array[]

[in][out] Array numérico para la puesta.

flag

[in] Dirección de indexación del array.

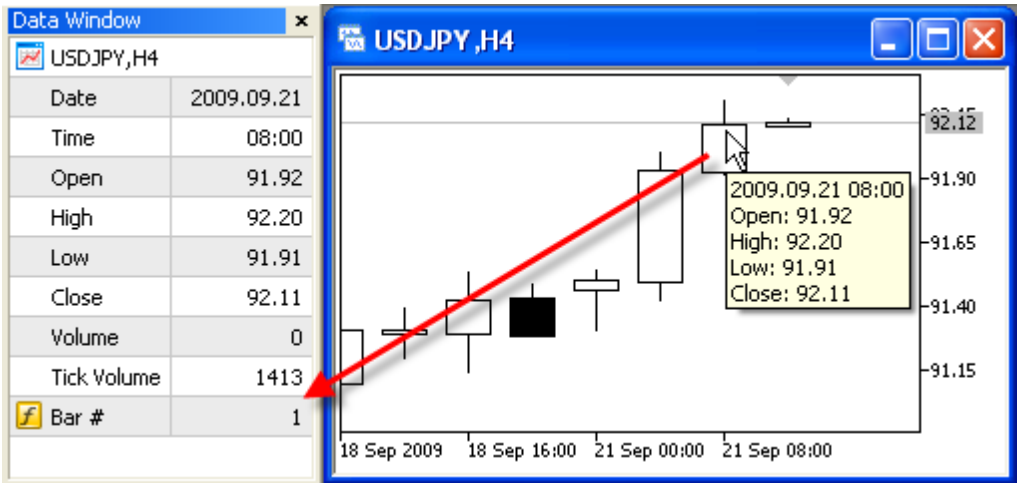
Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

No se puede poner la bandera AS_SERIES para los arrays multidimensionales y estáticos (es decir, los arrays cuyo tamaño ya está indicado en los corchetes en la etapa de compilación). La indexación en una serie temporal se diferencia de un array usual en que la indexación de los elementos de la serie temporal se realiza del fin del array al inicio (de los datos más recientes a los más antiguos).

Ejemplo: indicador que muestra el número de la barra



```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Numeration
#property indicator_label1  "Numeration"
#property indicator_type1   DRAW_LINE
#property indicator_color1  CLR_NONE
//--- indicator buffers
double         NumerationBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,NumerationBuffer,INDICATOR_DATA);
//--- establecemos la indexación para el buffer como en serie temporal
   ArraySetAsSeries(NumerationBuffer,true);
//--- establecemos la precisión de representación en DataWindow
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//--- como va a visualizarse el nombre del array de indicador en DataWindow
   PlotIndexSetString(0,PLOT_LABEL,"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[])
  {
//---  vamos a guardar la hora de apertura de la barra cero actual
   static datetime currentBarTimeOpen=0;
//--- revertimos el acceso al array time[] - lo hacemos como en serie temporal
   ArraySetAsSeries(time,true);
//--- Si la hora de barra cero se diferencia de la que estamos guardando,
   if(currentBarTimeOpen!=time[0])
     {
     //--- vamos a enumerar todos las barras desde el momento actual hacia dentro del
      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);
  }
```

Véase también

Acceso a las series temporales, ArrayGetAsSeries

ArraySize

Esta función devuelve el número de elementos del array especificado.

```
int  ArraySize(
   const void&  array[]    // array a comprobar
   );
```

Parámetros

array[]

[in] Array de cualquier tipo.

Valor devuelto

Valor del tipo int.

Nota

Para un array unidimensional el valor devuelto por la función ArraySize es igual al valor de ArrayRange(array,0).

Ejemplo:

```
void OnStart()
  {
//--- creación de arrays
   double one_dim[];
   double four_dim[][10][5][2];
//--- tamaños
   int one_dim_size=25;
   int reserve=20;
   int four_dim_size=5;
//--- variable auxiliar
   int size;
//--- adjudicamos memoria sin el backup
   ArrayResize(one_dim,one_dim_size);
   ArrayResize(four_dim,four_dim_size);
//--- 1. array unidimensional
   Print("+============================================================+");
   Print("Tamaños de arrays:");
   Print("1. Array unidimensional");
   size=ArraySize(one_dim);
   PrintFormat("Tamaño de la dimensión cero = %d, Tamaño del array = %d",one_dim_size
//--- 2. array multidimensional
   Print("2. Array multidimensional");
   size=ArraySize(four_dim);
   PrintFormat("Tamaño de la dimensión cero = %d, Tamaño del array = %d",four_dim_siz
//--- tamaños de dimensiones
   int d_1=ArrayRange(four_dim,1);
   int d_2=ArrayRange(four_dim,2);
   int d_3=ArrayRange(four_dim,3);
   Print("Prueba:");
   Print("Dimensión cero = Tamaño del array / (Primera dimensión * Segunda dimensión
   PrintFormat("%d = %d / (%d * %d * %d)",size/(d_1*d_2*d_3),size,d_1,d_2,d_3);
//--- 3. array unidimensional con memoria backup
   Print("3. Array unidimensional con memoria backup");
//--- aumentamos el valor 2 veces
   one_dim_size*=2;
//--- adjudicamos la memoria con backup
   ArrayResize(one_dim,one_dim_size,reserve);
//--- imprimimos el tamaño
   size=ArraySize(one_dim);
   PrintFormat("Tamaño con backup = %d, Tamaño real del array = %d",one_dim_size+rese
  }
```

ArraySort

Esta función clasifica un array numérico unidimensional en el orden ascendiente de izquierda a derecha.

```
bool  ArraySort(
   void&  array[]      // array para clasificar
   );
```

Parámetros

array[]

[in][out] Array numérico para ser clasificado.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

Un array marcado con la bandera AS_SERIES se clasifica en el orden descendiente.

Ejemplo:

```
#property description "El indicador analiza los datos del último mes y colorea todas
#property description "volúmenes de ticks grandes y pequeños. Para determinar estas v
#property description "array de los volúmenes de ticks. Las velas cuyos volúmenes se
#property description "por cientos del array se consideran pequeñas. Las velas cuyos
#property description "los últimos InpBigVolume por cientos del array se consideran g
//--- ajustes del indicador
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//--- plot
#property indicator_label1  "VolumeFactor"
#property indicator_type1   DRAW_COLOR_CANDLES
#property indicator_color1  clrDodgerBlue,clrOrange
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- constante predefinida
#define INDICATOR_EMPTY_VALUE 0.0
//--- parámetros de entrada
input int InpSmallVolume=15; // Porcentaje de volúmenes pequeños (<50)
input int InpBigVolume=20;   // Porcentaje de volúmenes grandes (<50)
//--- hora de inicio de análisis (va a desplazarse)
datetime ExtStartTime;
//--- búferes de indicadores
double   ExtOpenBuff[];
double   ExtHighBuff[];
double   ExtLowBuff[];
double   ExtCloseBuff[];
double   ExtColorBuff[];
//--- valores límite de volúmenes para visualización de velas
long     ExtLeftBorder=0;
long     ExtRightBorder=0;
//+------------------------------------------------------------------+
//| Recepción de valores de los límites para volúmenes de ticks      |
//+------------------------------------------------------------------+
bool GetVolumeBorders(void)
  {
//--- variables
   datetime stop_time;  // hora del fin de copiado
   long     buff[];     // búfer al que vamos a copiar
//--- hora final - hora actual
   stop_time=TimeCurrent();
//--- hora de inicio - un mes antes del tiempo actual
   ExtStartTime=GetStartTime(stop_time);
//--- obtenemos los valores de volúmenes de ticks
   ResetLastError();
   if(CopyTickVolume(Symbol(),Period(),ExtStartTime,stop_time,buff)==-1)
     {
      //--- no ha salido conseguir datos, devolvemos false para iniciar el comando de
      PrintFormat("No se ha podido conseguir los valores del volumen de ticks. Código
      return(false);
     }
//--- calculamos el tamaño del array
   int size=ArraySize(buff);
//--- clasificamos array
   ArraySort(buff);
//--- determinamos los valores del límite izquierdo y derecho para los volúmenes de t
   ExtLeftBorder=buff[size*InpSmallVolume/100];
   ExtRightBorder=buff[(size-1)*(100-InpBigVolume)/100];
//--- ejecución con éxito
   return(true);
  }
```

```
//+------------------------------------------------------------------+
//| Recepción de la fecha que sea un mes más antigua de la fecha que ha sido pasada
//+------------------------------------------------------------------+
datetime GetStartTime(const datetime stop_time)
  {
//--- convertimos el tiempo de finalización a la variable de la estructura del tipo M
   MqlDateTime temp;
   TimeToStruct(stop_time,temp);
//--- obtenemos la fecha que sea un mes más antigua
   if(temp.mon>1)
      temp.mon-=1;  // el mes en curso no es el primero de este año, entonces el nume
   else
     {
      temp.mon=12;  // el mes en curso es el primer mes del año, entonces el número d
      temp.year-=1; // y el número del año será menos uno
     }
//--- el número del día no va a superar 28
   if(temp.day>28)
      temp.day=28;
//--- devolvemos la fecha obtenida
   return(StructToTime(temp));
  }
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- comprobar si los parámetros de entrada satisfacen las condiciones
   if(InpSmallVolume<0 || InpSmallVolume>=50 || InpBigVolume<0 || InpBigVolume>=50)
     {
      Print("Parámetros de entrada incorrectos");
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- indicator buffers mapping
   SetIndexBuffer(0,ExtOpenBuff);
   SetIndexBuffer(1,ExtHighBuff);
   SetIndexBuffer(2,ExtLowBuff);
   SetIndexBuffer(3,ExtCloseBuff);
   SetIndexBuffer(4,ExtColorBuff,INDICATOR_COLOR_INDEX);
//--- establecemos el valor que no va a mostrarse
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,INDICATOR_EMPTY_VALUE);
//--- establecemos las etiquetas para los búferes de indicadores
   PlotIndexSetString(0,PLOT_LABEL,"Open;High;Low;Close");
//---
   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[])
  {
//--- comprobamos si hay más barras sin procesar
```

```
   if(prev_calculated<rates_total)
     {
      //--- obtenemos nuevos valores de los límites izquierdo y derecho para los volú
      if(!GetVolumeBorders())
         return(0);
     }
//--- variable de inicio para el cálculo de barras
   int start=prev_calculated;
//--- si los valores del indicador ya han sido calculados en el tick anterior, trabaj
   if(start>0)
      start--;
//--- establecemos la dirección directa de indexación en series temporales
   ArraySetAsSeries(time,false);
   ArraySetAsSeries(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
   ArraySetAsSeries(tick_volume,false);
//--- ciclo del cálculo de los valores del indicador
   for(int i=start;i<rates_total;i++)
     {
      //--- coloreamos las velas empezando de la fecha inicial
      if(ExtStartTime<=time[i])
        {
         //--- si el valor no es más bajo que el límite derecho, coloreamos la vela
         if(tick_volume[i]>=ExtRightBorder)
           {
            //--- obtenemos los datos para dibujar la vela
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- color DodgerBlue
            ExtColorBuff[i]=0;
            //--- seguimos con el ciclo
            continue;
           }
         //--- si el valor no es más alto que el límite izquierdo, coloreamos la vela
         if(tick_volume[i]<=ExtLeftBorder)
           {
            //--- obtenemos los datos para dibujar la vela
            ExtOpenBuff[i]=open[i];
            ExtHighBuff[i]=high[i];
            ExtLowBuff[i]=low[i];
            ExtCloseBuff[i]=close[i];
            //--- color Orange
            ExtColorBuff[i]=1;
            //--- seguimos con el ciclo
            continue;
           }
        }
      //--- para las barras que no han entrado en los cálculos, ponemos el valor vací
      ExtOpenBuff[i]=INDICATOR_EMPTY_VALUE;
      ExtHighBuff[i]=INDICATOR_EMPTY_VALUE;
      ExtLowBuff[i]=INDICATOR_EMPTY_VALUE;
      ExtCloseBuff[i]=INDICATOR_EMPTY_VALUE;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Conversión de datos

Se trata de un grupo de funciones encargadas de transformar datos de un formato al otro.

Sobre todo hay que destacar la función NormalizeDouble() que proporciona la precisión necesaria a la hora representar el precio. En las operaciones comerciales no se puede utilizar los precios no normalizados cuya precisión supera la requerida por el servidor comercial, aunque sea por un dígito.

Función	Acción
CharToString	Conversión del código del símbolo a una cadena de un carácter
DoubleToString	Conversión del valor numérico a una cadena de caracteres con una precisión especificada
EnumToString	Conversión del valor de una enumeración de cualquier tipo a una cadena de caracteres
NormalizeDouble	Redondeo de un número con punto flotante hasta una precisión especificada
StringToDouble	Conversión de una cadena que contiene la representación simbólica de un número a un número del tipo double
StringToInteger	Conversión de una cadena que contiene la representación simbólica de un número a un número del tipo int
StringToTime	Conversión de una cadena que contiene la hora y/o la fecha en el formato "yyyy.mm.dd [hh:mi]" al número del tipo datetime
TimeToString	Conversión del valor que contiene el tiempo en segundos transcurridos desde el 01.01.1970 a la cadena con el formato "yyyy.mm.dd hh:mi"
IntegerToString	Conversión del valor del tipo entero a una cadena de longitud definida
ShortToString	Conversión del código del símbolo (unicode) a una cadena de un carácter
ShortArrayToString	Copia una parte del array en una cadena
StringToShortArray	Copia caracteres de una cadena en una parte especificada de un array del tipo ushort
CharArrayToString	Conversión del código del símbolo (ansi) a una cadena de un carácter
StringToCharArray	Copia los caracteres de una cadena transformada de Unicode en ANSI en una parte seleccionada de un array del tipo uchar
ColorToARGB	Conversión del tipo color al tipo uint para

	conseguir la representación del color ARGB.
ColorToString	Conversión de valores de colores a una cadena del tipo "R,G,B"
StringToColor	Conversión de una cadena del tipo "R,G,B" o una cadena que contiene nombre de un color al valor del tipo color
StringFormat	Conversión de un número a una cadena conforme al formato especificado

Véase también

Uso de página de código

CharToString

Conversión del código del símbolo a una cadena de un carácter.

```
string  CharToString(
   uchar  char_code      // código numérico del símbolo
   );
```

Parámetros

char_code

[in] Código del símbolo ANSI.

Valor devuelto

Cadena que contiene un símbolo ANSI.

CharArrayToString

Copia y transforma una parte del array del tipo uchar en una cadena devuelta.

```
string  CharArrayToString(
   uchar    array[],             // array
   int      start=0,             // posición inicial en el array
   int      count=-1             // número de símbolos
   uint     codepage=CP_ACP      // página de código
   );
```

Parámetros

array[]

[in] Array del tipo uchar.

start=0

[in] Posición de que se empieza a copiar. Por defecto es 0.

count=-1

[in] Número de elementos del array a copiar. Determina la longitud de la cadena de resultado. Por defecto es -1, lo que significa el copiado hasta el final del array, o hasta encontrarse con el 0 de terminación.

codepage=CP_ACP

[in] Valor de la página de código. Para las páginas de código más usadas están previstas unas constantes correspondientes.

Valor devuelto

Una cadena.

Véase también

Uso de página de código

ColorToARGB

Esta función convierte el tipo color al tipo uint para conseguir la representación de color ARGB. El formato de color ARGB se utiliza durante la creación de recursos gráficos, visualización del texto y en la clase de la biblioteca estándar CCanvas.

```
uint  ColorToARGB(
   color  clr,          // color en el formato color para convertir
   uchar  alpha=255     // composición alfa que responde del grado de transparencia d
   );
```

Parámetros

clr

[in] Valor del color en la variable del tipo color.

alpha

[in] Valor de composición alfa para obtener el color en el formato ARGB. Se puede establecer el valor de 0 (el color del píxel puesto encima no cambia en absoluto la visualización del píxel del fondo) hasta 255 (el color se pone encima íntegramente y cubre totalmente el color del píxel del fondo). La transparencia del color en términos porcentuales se calcula según la siguiente fórmula (1-alpha/255)*100%. Es decir, cuanto menor sea el valor de la composición alfa, más transparente será el color.

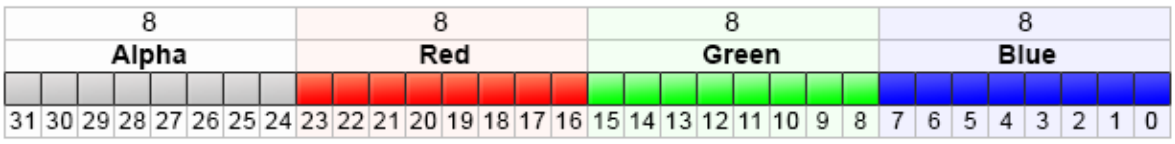
Valor devuelto

La representación del color en el formato ARGB, donde los valores Alfa, Red, Green, Blue (composición alfa, rojo, verde, azul) están escritos por orden en cuatro bytes del tipo uint.

Nota

RGB es el formato básico y mayoritariamente utilizado que sirve para describir el color del píxel sobre la pantalla en la computación gráfica. Los nombres de colores básicos se utilizan para establecer los componentes del color: rojo (Red), verde(Green) y azul (Blue). Cada componente se describe con un byte que establece la saturación de este color en un intervalo de 0 a 255 (de 0x00 a 0XFF en el formato hexadecimal). Puesto que el color blanco contiene todos los colores, se describe como 0xFFFFFF. O sea, cada uno de tres componentes está representado aquí con el valor máximo 0xFF.

Sin embargo, en algunas tareas se requiere la especificación de la transparencia del color con el fin de describir qué apariencia va a tener la imagen si la cubren con un color con cierto grado de transparencia. Para estas ocasiones se introduce el concepto de la composición alfa como un componente adicional al formato RGB. El esquema del formato ARGB se muestra en la figura de abajo.



Los valores ARGB habitualmente se indican en el formato hexadecimal, donde cada par de cifras representa por orden los valores de las composiciones Alpha, Red, Green y Blue. Por ejemplo, el color-ARGB 80FFFF00 representa el amarillo con opacidad de 50,2 %. Al principio va 0x80 que

establece el 50,2% de la composición alfa, ya que es un 50,2% del valor 0xFF. Luego el primer par FF representa el valor máximo del componente rojo; el siguiente par FF establece la misma intensidad del componente verde; y el último par 00 representa el mínimo valor del componente azul (ausencia del azul). La combinación del verde y el rojo produce el amarillo. Si la composición alfa no se utiliza, la entrada puede ser reducida a 6 dígitos RRGGBB. Por esa razón los valores de la composición alfa se guardan en los bits superiores del tipo de números enteros uint.

En función del contexto, los números hexadecimales pueden escribirse con el prefijo '0x' o '#', por ejemplo, 80FFFF00, o 0x80FFFF00, o #80FFFF00.

Ejemplo:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- establecemos la transparencia
   uchar alfa=0x55;  // el valor 0x55 significa 55/255=21,6 % de transparencia
   //--- obtenemos la conversión a ARGB para el color clrBlue
   PrintFormat("0x%.8X - clrBlue",clrBlue);
   PrintFormat("0x%.8X - clrBlue ARGB with alfa=0x55 (transparency 21.6%%)",ColorToAR
   //--- obtenemos la conversión a ARGB para el color clrGreen
   PrintFormat("0x%.8X - clrGreen",clrGreen);
   PrintFormat("0x%.8X - clrGreen ARGB with alfa=0x55 (transparency 21.6%%)",ColorToA
   //--- obtenemos la conversión a ARGB para el color clrRed
   PrintFormat("0x%.8X - clrRed",clrRed);
   PrintFormat("0x%.8X - clrRed ARGB with alfa=0x55 (transparency 21.6%%)",ColorToARG
  }
```

Véase también

Recursos, ResourceCreate(), TextOut(), Tipo color, Tipos char, short, int y long

ColorToString

Conversión de un valor de colores a una cadena del tipo "R,G,B".

```
string  ColorToString(
   color  color_value,     // valor de color
   bool   color_name       // mostrar nombre de color o no
   );
```

Parámetros

color_value

[in] Valor del color en la variable del tipo color.

color_name

[in] La señal de necesidad de devolver el nombre del color, si el valor del color coincide con una de las constantes de color predefinidas.

Valor devuelto

Representación literal de color como "R,G,B", donde R, G y B son constantes decimales de 0 a 255 convertidas a una cadena. Si el parámetro color_name=true está definido, se intenta convertir el valor del color al nombre del color.

Ejemplo:

```
   string clr=ColorToString(C'0,255,0'); // color verde
   Print(clr);

   clr=ColorToString(C'0,255,0',true);   // recibir constante de color
   Print(clr);
```

DoubleToString

Conversión del valor numérico a una cadena de caracteres.

```
string  DoubleToString(
   double  value,      // número
   int     digits=8    // número de dígitos después del punto decimal
   );
```

Parámetros

value

[in] Valor con punto flotante.

digits

[in] Formato de precisión. Si el valor digits se encuentra dentro del rango de 0 a 16, obtenemos la representación literal del número con la cantidad especificada de dígitos después del punto. Si el valor digits se encuentra dentro del rango de -1 a -16, obtenemos la representación literal del número en el formato científico con la cantidad especificada de dígitos después del punto. En todos los demás casos, el valor literal del número tendrá 8 dígitos después del punto.

Valor devuelto

Cadena que contiene representación de símbolos del número en el formato de precisión especificado.

Ejemplo:

```
   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,-16));
   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,-20));
```

Véase también

NormalizeDouble, StringToDouble

EnumToString

Convierte el valor de una enumeración de cualquier tipo a una cadena de caracteres.

```
string  EnumToString(
   any_enum  value      // valor de la enumeración de cualquier tipo
   );
```

Parámetros

value

[in] Valor de la enumeración de cualquier tipo.

Valor devuelto

Una cadena que contiene la representación de texto del valor. Para obtener la información sobre el error, hay que llamara a la función GetLastError().

Nota

La función GetLastError() puede establecer los siguientes valores en la variable _LastError:

- ERR_INTERNAL_ERROR - error del entorno de ejecución
- ERR_NOT_ENOUGH_MEMORY - no hay suficiente memoria para completar la operación
- ERR_INVALID_PARAMETER - no se puede aceptar el nombre del valor de la enumeración

Ejemplo:

```
enum interval  // enumeración de constantes nombradas
  {
   month=1,     // período de un mes
   two_months,  // dos meses
   quarter,     // tres meses - trimestre
   halfyear=6,  // semestre
   year=12,     // año - 12 meses
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- se establece el período de tiempo que es igual a un mes
   interval period=month;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- se establece el período de tiempo que es igual a un trimestre (tres meses))
   period=quarter;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- se establece el período de tiempo que es igual a un año (12 meses))
   period=year;
   Print(EnumToString(period)+"="+IntegerToString(period));

//--- comprobamos cómo se muestra el tipo de la orden
   ENUM_ORDER_TYPE type=ORDER_TYPE_BUY;
   Print(EnumToString(type)+"="+IntegerToString(type));

//--- comprobamos cómo se muestra el valor incorrecto
   type=WRONG_VALUE;
   Print(EnumToString(type)+"="+IntegerToString(type));

// Resultado de ejecución:
// month=1
// quarter=3
// year=12
// ORDER_TYPE_BUY=0
// ENUM_ORDER_TYPE::-1=-1
  }
```

Véase también

Enumeraciones, Variables de entrada Input

IntegerToString

Convierte el valor del tipo entero a una cadena de longitud especificada y devuelve la cadena obtenida.

```
string  IntegerToString(
   long     number,              // número
   int      str_len=0,           // longitud de la cadena en la salida
   ushort   fill_symbol=' '      // relleno
   );
```

Parámetros

number

[in] Número para conversión.

str_len=0

[in] Longitud de la cadena. Si la longitud de la cadena obtenida resulta ser más de la especificada, la cadena no se recorta. Si la longitud de la cadena obtenida resulta ser menos de la especificada, los símbolos de relleno se añadirán a esta cadena por la izquierda.

fill_symbol=' '

[in] Símbolo de relleno. Por defecto es un espacio.

Valor devuelto

Cadena.

ShortToString

Convierte el código del símbolo (unicode) a una cadena de un carácter y devuelve la cadena obtenida.

```
string  ShortToString(
   ushort  symbol_code      // símbolo
   );
```

Parámetros

symbol_code

[in] Código del símbolo. En vez del código del símbolo se puede usar una cadena literal que contiene un símbolo, o una cadena literal con un código hexadecimal de dos bytes que corresponde al código de la tabla Unicode.

Valor devuelto

Cadena.

ShortArrayToString

Copia una parte del array en la cadena devuelta.

```
string  ShortArrayToString(
   ushort  array[],      // array
   int     start=0,      // posición inicial en el array
   int     count=-1      // cantidad de símbolos
   );
```

Parámetros

array[]

[in] Array del tipo ushort (análogo del tipo wchar_t).

start=0

[in] Punto de partida del copiado. Por defecto es 0.

count=-1

[in] Número de elementos del array para ser copiados. Determina la longitud de la cadena resultante. El valor por defecto es -1, esto significa el copiado hasta el final del array, o hasta el 0 de terminación.

Valor devuelto

Cadena.

TimeToString

Conversión del valor que contiene el tiempo en segundos transcurridos desde el 01.01.1970 a la cadena con el formato "yyyy.mm.dd hh:mi".

```
string  TimeToString(
   datetime  value,                          // número
   int       mode=TIME_DATE|TIME_MINUTES     // formato output
   );
```

Parámetros

value

[in] Tiempo en segundos de 00:00 1 de Enero de 1970.

mode=TIME_DATE|TIME_MINUTES

[in] Modo adicional del output de datos. Puede ser una bandera o bandera combinada:
TIME_DATE obtiene resultado en formato " yyyy.mm.dd " ,
TIME_MINUTES obtiene resultado en formato " hh:mm " ,
TIME_SECONDS obtiene resultado en formato " hh:mm:ss ".

Valor devuelto

Una cadena.

NormalizeDouble

Redondeo del número con punto flotante hasta una precisión especificada.

```
double  NormalizeDouble(
   double  value,      // número normalizado
   int     digits      // cantidad de símbolos después del punto decimal
   );
```

Parámetros

value

[in] Valor con punto flotante.

digits

[in] Formato de precisión, número de dígitos después del punto decimal (0-8).

Valor devuelto

Valor del tipo double con una precisión especificada.

Nota

Valores calculados StopLoss, TakeProfit y los valores de precio de apertura de pedidos pendientes tienen que ser normalizados con la precisión cuyo valor puede ser obtenido por la función Digits().

Hay que tener en cuenta que cuando un número normalizado se visualiza en el Diario mediante Print (), éste puede tener una cantidad de dígitos tras la coma más grande de lo esperado. Por ejemplo,

```
   double a=76.671;              // número normalizado con 3 dígitos tras la coma
   Print("Print(76.671)=",a);    // vamos a mostrarlo tal como es
   Print("DoubleToString(a,8)=",DoubleToString(a,8)); // vamos a mostrarlo con una pr
```

mostrará en el terminal:

```
   DoubleToString(a,8)=76.67100000
   Print(76.671)=76.67100000000001
```

Ejemplo:

```
   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));
/*
   Resultado:
   pi= 3.1415926535897931
   NormalizeDouble(pi,3)= 3.1419999999999999
   NormalizeDouble(pi,8)= 3.1415926499999998
   NormalizeDouble(pi,0)= 3.0000000000000000
*/
```

Véase también

DoubleToString, Tipos reales (double, float), Conversión de tipos,

StringToCharArray

Copia los caracteres de una cadena transformada de Unicode en ANSI en una parte indicada de un array del tipo uchar. La función devuelve el número de elementos copiados.

```
int  StringToCharArray(
   string  text_string,          // cadena de origen
   uchar&  array[],              // array
   int     start=0,              // posición de inicio en el array
   int     count=-1              // cantidad de símbolos
   uint    codepage=CP_ACP       // página de códigos
   );
```

Parámetros

text_string

[in] Cadena para el copiado.

array[]

[out] Array del tipo uchar.

start=0

[in] Posición de donde se empieza a copiar. Por defecto es 0.

count=-1

[in] Número de elementos del array para ser copiados. Determina la longitud de la cadena resultante. El valor por defecto es -1, esto significa el copiado hasta el final del array, o hasta el 0 de terminación. El 0 de terminación también va a ser copiado en el array de destino; en este caso, si hace falta, el tamaño del array dinámico puede ser aumentado hasta el tamaño de la cadena. Si el tamaño del array dinámico supera la longitud de la cadena, el tamaño del array no será reducido.

codepage=CP_ACP

[in] Valor de la página de códigos. Para algunas páginas de códigos más usadas están previstas constantes correspondientes.

Valor devuelto

Número de elementos copiados.

Véase también

Uso de página de código

StringToColor

Convierte una cadena del tipo "R,G,B" o una cadena que contiene nombre de un color al valor del tipo color.

```
color  StringToColor(
   string  color_string      // representación literal de color
   );
```

Parámetros

color_string

[in] Representación literal de un color del tipo "R,G,B" o nombre de uno de los colores_Web predefinidos.

Valor devuelto

Valor del color.

Ejemplo:

```
   color str_color=StringToColor("0,127,0");
   Print(str_color);
   Print((string)str_color);
//--- cambiamos un poco el color
   str_color=StringToColor("0,128,0");
   Print(str_color);
   Print((string)str_color);
```

StringToDouble

La función convierte la cadena que contiene la representación simbólica de un número al número del tipo double.

```
double  StringToDouble(
   string  value      // cadena
   );
```

Parámetros

value

[in] Cadena que contiene la representación simbólica de un número.

Valor devuelto

Valor del tipo double.

StringToInteger

Conversión de una cadena que contiene la representación simbólica de un número al número del tipo int (entero).

```
long  StringToInteger(
   string  value      // Cadena
   );
```

Parámetros

value

[in] Cadena que contiene el número.

Valor devuelto

Valor del tipo long.

StringToShortArray

Copia una cadena símbolo por símbolo en una parte indicada de un array del tipo ushort. La función devuelve el número de elementos copiados.

```
int  StringToShortArray(
   string  text_string,      // cadena de origen
   ushort& array[],          // array
   int     start=0,          // posición de inicio en el array
   int     count=-1          // cantidad de símbolos
   );
```

Parámetros

text_string

[in] Cadena para el copiado.

array[]

[out] Array del tipo ushort (análogo del tipo wchar_t).

start=0

[in] Posición de donde se empieza a copiar. Por defecto es 0.

count=-1

[in] Número de elementos del array para ser copiados. Determina la longitud de la cadena resultante. El valor por defecto es -1, esto significa el copiado hasta el final del array, o hasta el 0 de terminación. El 0 de terminación también va a ser copiado en el array de destino; en este caso, si hace falta, el tamaño del array dinámico puede ser aumentado hasta el tamaño de la cadena. Si el tamaño del array dinámico supera la longitud de la cadena, el tamaño del array no será reducido.

Valor devuelto

Número de elementos copiados.

StringToTime

Conversión de una cadena que contiene la hora y/o la fecha en el formato "yyyy.mm.dd [hh:mi]" al número del tipo datetime.

```
datetime  StringToTime(
   string  value      // cadena de fecha
   );
```

Parámetros

value

[in] Cadena en el formato " yyyy.mm.dd hh:mi ".

Valor devuelto

Valor del tipo datetime que contiene la cantidad de segundos transcurridos desde 01.01.1970.

StringFormat

Formatea los parámetros obtenidos y devuelve una cadena.

```
string  StringFormat(
   string  format,     // cadena con descripción de formato
   ...     ...         // parámetros
   );
```

Parámetros

format

[in] Cadena que contiene el modo de formatear. Las reglas de formatear son las mismas que para la función PrintFormat

...

[in] Parámetros separados por coma.

Valor devuelto

Cadena.

Véase también

PrintFormat, DoubleToString,ColorToString, TimeToString

Funciones matemáticas

Conjunto de funciones matemáticas y trigonométricas.

Función	Acción
MathAbs	Devuelve el valor absoluto (modular) de un número que se le ha pasado
MathArccos	Devuelve el valor de arcocoseno de (x) en radianes
MathArcsin	Devuelve el valor de arcseno de (x) en radianes
MathArctan	Devuelve el valor de arcotangete de (x) en radianes
MathCeil	Devuelve el valor numérico entero más cercano desde arriba
MathCos	Devuelve el coseno de un número
MathExp	Devuelve el exponente de un número
MathFloor	Devuelve el valor numérico entero más cercano desde abajo
MathLog	Devuelve un logaritmo neperiano (natural)
MathMax	Devuelve el valor máximo de dos valores numéricos
MathMin	Devuelve el valor mínimo de dos valores numéricos
MathMod	Devuelve el resto real de la división de dos números
MathPow	Eleva la base a la potencia indicada
MathRand	Devuelve un número pseudoaleatorio en el rango de 0 a 32767
MathRound	Redondea un número hasta el entero más cercano
MathSin	Devuelve el seno de un número
MathSqrt	Devuelve una raíz cuadrada
MathSrand	Define el estado inicial del generador pseudoaleatorio de números enteros
MathTan	Devuelve la tangete de un número
MathIsValidNumber	Verifica la correctitud de un número real

MathAbs

La función devuelve el valor absoluto (modular) de un número que se le ha pasado.

```
double  MathAbs(
   double  value      // número
   );
```

Parámetros

value

[in] Valor numérico.

Valor devuelto

Valor del tipo double, más o igual a cero.

Nota

En vez de la función MathAbs() se puede usar la función fabs().

MathArccos

Devuelve el valor de arcocoseno de (x) en el rango de 0 a π en radianes.

```
double  MathArccos(
   double  val     // -1<val<1
   );
```

Parámetros

val

[in] Valor val entre -1 y 1, cuyo arcocoseno tiene que ser calculado.

Valor devuelto

Arcocoseno de un número en radianes. Si val es menos de -1 o más de 1, la función devuelve NaN (valor indeterminado).

Nota

En vez de la función MathArccos() se puede usar la función acos().

Véase también

Tipos reales (double, float)

MathArcsin

Devuelve arcseno (x) en el rango de -π/2 a π/2 radianes.

```
double  MathArcsin(
   double  val      // -1<value<1
   );
```

Parámetros

val

[in] Valor val entre -1 y 1, cuyo arcseno tiene que ser calculado.

Valor devuelto

Arcseno de un número val en radianes en el rango de -π/2 a π/2 radianes. Si val es menos de -1 o más de 1, la función devuelve NaN (valor indeterminado).

Nota

En vez de la función MathArcsin() se puede usar la función asin().

Véase también

Tipos reales (double, float)

MathArctan

Devuelve arcotangete de (x). Si x es igual a 0, la función devuelve 0.

```
double  MathArctan(
   double  value      // tangete
   );
```

Parámetros

value

[in] Número que representa tangete.

Valor devuelto

MathArctan devuelve un valor en el rango de -π/2 a π/2 radianes.

Nota

En vez de la función MathArctan() se puede usar la función atan().

MathCeil

Devuelve el valor numérico entero más cercano desde arriba.

```
double  MathCeil(
   double  val      // número
   );
```

Parámetros

val

[in] Valor numérico.

Valor devuelto

Valor numérico que representa el número entero más pequeño que supera o equivale a val.

Nota

En vez de la función MathCeil() se puede usar la función ceil().

MathCos

La función devuelve el coseno de un ángulo.

```
double  MathCos(
   double  value      // número
   );
```

Parámetros

value

[in] Ángulo en radianes.

Valor devuelto

Valor del tipo double en el rango de -1 a 1.

Nota

En vez de la función MathCos() se puede usar la función cos().

MathExp

La función devuelve el valor del número e elevado a la potencia d.

```
double  MathExp(
   double  value      // potencia para el número e
   );
```

Parámetros

value

[in] Número que especifica la potencia.

Valor devuelto

Número del tipo double. Al superar el límite la función devuelve INF (infinito), en caso de perder el orden MathExp devolverá 0.

Nota

En vez de la función MathExp() se puede usar la función exp().

Véase también

Tipos reales (double, float)

MathFloor

Devuelve el valor numérico entero más cercano desde abajo.

```
double  MathFloor(
   double  val     // número
   );
```

Parámetros

val

[in] Valor numérico.

Valor devuelto

Valor numérico que representa el número entero más grande, que es menos o igual a val.

Nota

En vez de la función MathFloor() se puede usar la función floor().

MathLog

Devuelve un logaritmo neperiano (natural).

```
double  MathLog(
   double  val      // número para coger el logaritmo
   );
```

Parámetros

val

[in] Valor cuyo logaritmo tiene que ser calculado.

Valor devuelto

En caso de éxito devuelve el logaritmo natural de val. Si el valor de val es negativo, la función devuelve NaN (valor indeterminado). Si val es igual a 0, la función devuelve INF (infinito).

Nota

En vez de la función MathLog() se puede usar la función log().

Véase también

Tipos reales (double, float)

MathLog

Devuelve el logaritmo de un número en base 10.

```
double  MathLog10(
   double  val      // número para coger el logaritmo
   );
```

Parámetros

val

[in] Valor cuyo logaritmo decimal tiene que ser calculado.

Valor devuelto

En caso de éxito devuelve el logaritmo decimal de val. Si el valor de val es negativo, la función devuelve NaN (valor indeterminado). Si val es igual a 0, la función devuelve INF (infinito).

Nota

En vez de la función MathLog10() se puede usar la función log10().

Véase también

Tipos reales (double, float)

MathMax

La función devuelve el valor máximo de dos valores numéricos.

```
double  MathMax(
   double  value1,      // primer número
   double  value2       // segundo número
   );
```

Parámetros

value1

[in] Primer valor numérico.

value2

[in] Segundo valor numérico.

Valor devuelto

El número más grande de los dos.

Nota

En vez de la función MathMax() se puede usar la función fmax(). Las funciones fmax(), fmin(), MathMax(), MathMin() pueden trabajar con tipos enteros sin conversión al tipo double.

Si los parámetros de diferentes tipos se pasan a la función, el parámetro del tipo menor automáticamente se convierte al tipo mayor. El tipo de valor devuelto corresponde al tipo mayor.

Si se pasan los datos del mismo tipo, no se realiza ninguna conversión.

MathMin

La función devuelve el valor mínimo de dos valores numéricos.

```
double  MathMin(
   double  value1,      // primer número
   double  value2       // segundo número
   );
```

Parámetros

value1

[in] Primer valor numérico.

value2

[in] Segundo valor numérico.

Valor devuelto

El número más pequeño de los dos.

Nota

En vez de la función MathMin() se puede usar la función fmin(). Las funciones fmax(), fmin(), MathMax(), MathMin() pueden trabajar con tipos enteros sin conversión al tipo double.

Si los parámetros de diferentes tipos se pasan a la función, el parámetro del tipo menor automáticamente se convierte al tipo mayor. El tipo de valor devuelto corresponde al tipo mayor.

Si se pasan los datos del mismo tipo, no se realiza ninguna conversión.

MathMod

Devuelve el resto real de la división de dos números.

```
double  MathMod(
   double  value,       // dividiendo
   double  value2       // divisor
   );
```

Parámetros

value

[in] Valor del dividiendo.

value2

[in] Valor del divisor.

Valor devuelto

La función MathMod calcula el resto real f de val / y de tal manera que val = i * y + f , donde i es un número entero, f tiene el mismo signo que val, y el valor absoluto de f es menos que el valor absoluto de y.

Nota

En vez de la función MathMod() se puede usar la función fmod().

MathPow

Eleva una base a una potencia indicada.

```
double  MathPow(
   double  base,          // base
   double  exponent       // exponente
   );
```

Parámetros

base

[in] Base.

exponent

[in] Valor de potenciación.

Valor devuelto

Valor de la base elevada a la potencia indicada.

Nota

En vez de la función MathPow() se puede usar la función pow().

MathRand

Devuelve un número pseudoaleatorio en el rango de 0 a 32767.

```
int  MathRand();
```

Valor devuelto

Número entero en el rango de 0 a 32767.

Nota

Antes de la primera llamada a la función hay que usar la función MathSrand, con el fin de poner el generador pseudoaleatorio de números en su posición inicial.

Nota

En vez de la función MathRand() se puede usar la función rand().

MathRound

Devuelve un valor redondeado hasta el número entero más cercano del valor numérico especificado.

```
double  MathRound(
   double  value      // valor a redondear
   );
```

Parámetros

value

[in] Valor numérico para ser redondeado.

Valor devuelto

Valor redondeado hasta el número entero más cercano.

Nota

En vez de la función MathRound() se puede usar la función round().

MathSin

Devuelve el seno de un ángulo especificado.

```
double  MathSin(
   double  value      // argumento en radianes
   );
```

Parámetros

value

[in] Ángulo en radianes.

Valor devuelto

Seno de un ángulo indicado en radianes. Devuelve valor en el rango de -1 a 1.

Nota

En vez de la función MathSin() se puede usar la función sin().

MathSqrt

Devuelve la raíz cuadrada de un número.

```
double  MathSqrt(
   double  value      // número positivo
   );
```

Parámetros

value

[in] Valor numérico positivo.

Valor devuelto

Raíz cuadrada de value. Si value es negativo, MathSqrt devuelve NaN (valor indeterminado).

Nota

En vez de la función MathSqrt() se puede usar la función sqrt().

Véase también

Tipos reales (double, float)

MathSrand

Establece el estado inicial para generar una serie de números enteros pseudoaleatorios.

```
void  MathSrand(
   int  seed      // número de inicialización
   );
```

Parámetros

seed

[in] Número inicial para una fila de números aleatorios.

Valor devuelto

No hay valor devuelto.

Nota

La función MathRand() sirve para generar una secuencia de números pseudoaleatorios. La llamada de MathSrand() con un cierto número inicializador permite recibir siempre la misma secuencia de números pseudoaleatorios.

Para garantizar la recepción de una secuencia irrepetible, utilice la llamada de MathSrand (GetTickCount()), ya que el valor GetTickCount() se aumenta desde el arranque del sistema operativo y no se repite durante 49 días hasta que se llene el contador incorporado de milisegundos. El uso de MathSrand(TimeCurrent()) no vale porque la función TimeCurrent() devuelve la hora de llegada del último tick que puede ser intacto durante bastante tiempo, por ejemplo durante el fin de semana.

La inicialización del generador de números pseudoaleatorios con el uso de MathSrand() para los indicadores y EAs es mejor hacer dentro del manejador OnInit(). Esto permite evitar los posteriores numerosos arranques del generador en OnTick() y OnCalculate().

En vez de la función MathSrand() se puede usar la función srand().

Ejemplo:

```
#property description "El indicador demuestra el teorema del límite central que indic
#property description "La suma de un número suficientemente grande de variables aleat
#property description "que tienen aproximadamente las mismas dimensiones (ninguno de
#property description "sin ejercer ninguna contribución determinante en la suma), «se

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- propiedades de construcción gráfica
#property indicator_label1  "Label"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRoyalBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  5
//--- variable input
input int      sample_number=10;
//--- búfer indicador para dibujar la distribución
double         LabelBuffer[];
//--- contador de ticks
double         ticks_counter;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- enlace del array con el búfer de indicador
   SetIndexBuffer(0,LabelBuffer,INDICATOR_DATA);
//--- damos la vuelta al búfer de indicador desde el presente hasta el pasado
   ArraySetAsSeries(LabelBuffer,true);
//--- inicializamos el generador de números aleatorios
   MathSrand(GetTickCount());
//--- inicializamos el contador de 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[])
  {
//--- con el contador cero ponemos el búfer de indicador a cero
   if(ticks_counter==0) ArrayInitialize(LabelBuffer,0);
//--- aumentamos el contador
   ticks_counter++;
//--- hace falta poner a cero periódicamente el contador de ticks para reavivar la di
   if(ticks_counter>100)
     {
      Print("Tras poner a cero los valores del indicador, volvemos a rellenar las cel
      ticks_counter=0;
     }
//--- obtenemos la muestra de valores aleatorios como la suma de tres números de 0 a
   for(int i=0;i<sample_number;i++)
     {
```

```
      //--- cálculo del índice de la celda en la que se coloca el número aleatorio co
      int rand_index=0;
      //--- obtenemos tres números aleatorios de 0 a 7
      for(int k=0;k<3;k++)
        {
         //--- el resto de la división por 7 devolverá un valor de 0 a 6
         rand_index+=MathRand()%7;
        }
      //--- aumentamos a uno el valor en la celda con el número rand_index
      LabelBuffer[rand_index]++;
     }
//--- salimos del manejador OnCalculate()
   return(rates_total);
  }
```

MathTan

Devuelve la tangete de un número.

```
double  MathTan(
   double  rad      // argumento en radianes
   );
```

Parámetros

rad

[in] Ángulo en radianes.

Valor devuelto

Tangete del número rad. Si rad es más de o igual a 263 o menos de o igual a -263, ocurre la perdida del valor y la función devuelve un número indeterminado.

Nota

En vez de la función MathTan() se puede usar la función tan().

Véase también

Tipos reales (double, float)

MathIsValidNumber

Verifica la correctitud de un número real.

```
bool  MathIsValidNumber(
   double  number      // número a comprobar
   );
```

Parámetros

number

[in] Número a comprobar.

Valor devuelto

Devuelve true, si el valor que se comprueba es un número real aceptable. Si el valor que se comprueba es más o menos infinito, o se trata de "no número" (NaN - not a number), la función devuelve false.

Ejemplo:

```
   double abnormal=MathArcsin(2.0);
   if(!MathIsValidNumber(abnormal)) Print("Atención! MathArcsin(2.0) =",abnormal);
```

Véase también

Tipos reales (double, float)

Funciones literales

Es un grupo de funciones que operan con los datos del tipo string.

Función	Acción
StringAdd	Añade una subcadena especificada a una cadena en el lugar
StringBufferLen	Devuelve el tamaño del buffer distribuido para una cadena
StringCompare	Compara dos cadenas de caracteres, y devuelve 1 si la primera cadena es más grande que la segunda, devuelve 0 si las cadenas son iguales, y -1 (menos 1) si la primera es menor que la segunda.
StringConcatenate	Forma una cadena con los parámetros pasados
StringFill	Rellena una cadena especificada con símbolos especificados en el lugar
StringFind	Busca una subcadena en una cadena
StringGetCharacter	Devuelve el valor de un símbolo que se encuentra en la posición especificada de una cadena
StringInit	Inicializa una cadena con símbolos especificados y proporciona la longitud especificada de una cadena
StringLen	Devuelve el número de símbolos de una cadena
StringReplace	Reemplaza todas las subcadenas encontradas en una cadena de caracteres por una secuencia de símbolos.
StringSetCharacter	Devuelve copia de una cadena con el valor del símbolo modificado en una posición especificada
StringSplit	Obtiene las subcadenas por un separador establecido desde la cadena especificada y devuelve el número de subcadenas obtenidas
StringSubstr	Extrae una subcadena de una cadena de texto que se empieza desde una posición especificada
StringToLower	Transforma todos los símbolos de una cadena indicada en minúsculas en el lugar
StringToUpper	Transforma todos los símbolos de una cadena indicada en mayúsculas en el lugar

StringTrimLeft	Borra los símbolos de salto de línea, espacios y símbolos de tabulación en la parte izquierda de la cadena
StringTrimRight	Borra los símbolos de salto de línea, espacios y símbolos de tabulación en la parte derecha de la cadena

StringAdd

La función añade una subcadena especificada al final de una cadena en el lugar.

```
bool  StringAdd(
   string&  string_var,        // cadena a la que añadimos
   string   add_substring      // cadena que se añade
   );
```

Parámetros

string_var

[in][out] Cadena que va a ser completada con otra.

add_substring

[in] Cadena que va a ser añadida al final de la cadena fuente.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false. Para obtener el código de error hay que llamar a la función GetLastError().

Ejemplo:

```
void OnStart()
  {
   string a="a",b="b",c;
//--- primer método
   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);

//--- segundo método
   start=GetTickCount();
   for(i=0;i<length;i++)
     {
      StringAdd(a,b);
     }
   stop=GetTickCount();
   Print("time for 'StringAdd(a,b)' = ",(stop-start)," milliseconds, i = ",i);

//--- tercer método
   start=GetTickCount();
   a="a"; // volvemos a inicializar la variable a
   for(i=0;i<length;i++)
     {
```

```
      int k=StringConcatenate(c,a,b);
     }
   stop=GetTickCount();
   Print("time for 'StringConcatenate(c,a,b)' = ",(stop-start),"milliseconds, i = ",i
  }
```

Véase también

StringConcatenate

StringBufferLen

Devuelve el tamaño del buffer distribuido para una cadena.

```
int  StringBufferLen(
   string  string_var      // cadena
   )
```

Parámetros

string_var

[in] Cadena.

Valor devuelto

El valor 0 significa que esta cadena es una cadena constante y no se puede cambiar el contenido del buffer. -1 significa que la cadena pertenece al terminal de cliente y el cambio del contenido del buffer puede suponer unos resultados indeterminados.

Nota

El tamaño mínimo del buffer es igual a 16.

Ejemplo:

```
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));
  }
```

Véase también

StringAdd, StringInit, StringLen, StringFill

StringCompare

Esta función compara dos cadenas de caracteres y devuelve el resultado de la comparación en forma de un número entero.

```
int  StringCompare(
   const string&  string1,                // la primera cadena a comparar
   const string&  string2,                // la segunda cadena a comparar
   bool           case_sensitive=true     // modo de sensibilidad a mayúsculas
   );
```

Parámetros

string1

[in] La primera cadena.

string2

[in] La segunda cadena.

case_sensitive=true

[in] Selección del modo de sensibilidad a mayúsculas. Si es true, entonces "A">"a". Si es false, entonces "A"="a". Por defecto, el valor de este parámetro es igual a true.

Valor devuelto

- -1 (menos uno), si string1<string2
- 0 (cero), si string1=string2
- 1 (uno), si string1>string2

Nota

Las cadenas se comparan carácter por carácter. Los caracteres se comparan en orden alfabético de acuerdo con la página de código actual.

Ejemplo:

```
void OnStart()
  {
//--- ¿Qué es más grande, una manzana o una casa?
   string s1="Apple";
   string s2="home";

//--- modo de sensibilidad a mayúsculas activado
   int result1=StringCompare(s1,s2);
   if(result1>0) PrintFormat("Comparación sensible a mayúsculas: %s > %s",s1,s2);
   else
     {
      if(result1<0)PrintFormat("Comparación sensible a mayúsculas: %s < %s",s1,s2);
      else PrintFormat("Comparación sensible a mayúsculas: %s = %s",s1,s2);
     }

//--- modo de sensibilidad a mayúsculas desactivado
   int result2=StringCompare(s1,s2,false);
   if(result2>0) PrintFormat("Comparación no sensible a mayúsculas: %s > %s",s1,s2);
   else
     {
      if(result2<0)PrintFormat("Comparación no sensible a mayúsculas: %s < %s",s1,s2)
      else PrintFormat("Comparación no sensible a mayúsculas: %s = %s",s1,s2);
     }
/* Resultado:
     Comparación sensible a mayúsculas: Apple < home
     Comparación no sensible a mayúsculas: Apple < home
*/
  }
```

Véase también

Tipo string, CharToString(), ShortToString(), StringToCharArray(), StringToShortArray(), StringGetCharacter(), Uso de página de código

StringConcatenate

Forma una cadena con los parámetros pasados y devuelve el tamaño de la cadena formada. Los parámetros pueden ser de cualquier tipo. El número de parámetros no puede ser menos de 2 y más de 64.

```
int  StringConcatenate(
   string&  string_var,    // cadena para formar
   void argument1          // primer parámetro de cualquier tipo simple
   void argument2          // segundo parámetro de cualquier tipo simple
   ...                     // siguiente parámetro de cualquier tipo simple
   );
```

Parámetros

string_var

[out] Cadena que va a ser formada como resultado de concatenación.

argumentN

[in] Cualquieres valores separados por comas. De 2 a 63 parámetros de cualquier tipo simple.

Valor devuelto

Devuelve la longitud de la cadena formada por medio de la concatenación de parámetros transformados en el tipo string. Los parámetros se transforman en las cadenas según las mismas reglas que en las funciones Print() y Comment().

Véase también

StringAdd

StringFill

Llena una cadena especificada con símbolos especificados en el lugar.

```
bool  StringFill(
   string&   string_var,       // cadena para llenar
   ushort    character         // símbolos que van a llenar la cadena
   );
```

Parámetros

string_var

[in][out] Cadena que va a ser llenada con símbolos especificados.

character

[in] Símbolo con el que va a ser llenada la cadena.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false. Para obtener el código de error hay que llamar a la función GetLastError().

Nota

Llenar una cadena en el lugar significa que los símbolos se insertan directamente en la cadena sin operaciones intermedias de creación de unas nuevas cadenas y sin copiado. Esto permite ahorrar el tiempo de trabajo con una cadena en dicha función.

Ejemplo:

```
void OnStart()
  {
   string str;
   StringInit(str,20,'_');
   Print("str = ",str);
   StringFill(str,0);
   Print("str = ",str,": StringBufferLen(str) = ", StringBufferLen(str));
  }
// Resultado
//   str = ____________________
//   str =  : StringBufferLen(str) = 20
//
```

Véase también

StringBufferLen, StringLen, StringInit

StringFind

Busca una subcadena en una cadena.

```
int  StringFind(
   string  string_value,        // cadena en la que buscamos
   string  match_substring,     // lo que buscamos
   int     start_pos=0          // punto de partida de la búsqueda
   );
```

Parámetros

string_value

[in] Cadena en la que se realiza la búsqueda.

match_substring

[in] Subcadena buscada.

start_pos=0

[in] Posición en la cadena desde la cual debe empezarse la búsqueda.

Valor devuelto

Devuelve el número de posición en la cadena desde la cual se empieza la subcadena buscada, o devuelve -1 si la subcadena no ha sido encontrada.

StringGetCharacter

Devuelve el valor de un símbolo que se encuentra en la posición especificada de una cadena.

```
ushort  StringGetCharacter(
   string  string_value,     // cadena
   int     pos               // posición del símbolo en la cadena
   );
```

Parámetros

string_value

[in] Cadena.

pos

[in] Posición del símbolo en la cadena. Puede ser de 0 a StringLen(text) -1.

Valor devuelto

Código del símbolo, o en caso de algún error devuelve 0. Para obtener el código de error hay que llamar a la función GetLastError().

StringInit

Inicializa una cadena con símbolos especificados y proporciona la longitud especificada de una cadena.

```
bool  StringInit(
   string&   string_var,       // cadena para inicialización
   int       new_len=0,        // longitud necesaria después de inicialización
   ushort    character=0       // símbolo con el que se llena la cadena
   );
```

Parámetros

string_var

[in][out] Cadena que tiene que ser inicializada o deinicializada.

new_len=0

[in] Longitud de la cadena después de inicialización. Si la longitud=0, la cadena se deinicializa, es decir, el buffer de la cadena se limpia y la dirección del buffer se pone a cero.

character=0

[in] Símbolo para llenar la cadena.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false. Para obtener el código de error hay que llamar a la función GetLastError().

Nota

Si character=0 y la longitud new_len>0, entonces el buffer de la cadena con longitud indicada será distribuido y llenado con ceros. La longitud de la cadena será igual a cero porque el buffer entero está llenado con terminadores de la cadena.

Ejemplo:

```
void OnStart()
  {
//---
   string str;
   StringInit(str,200,0);
   Print("str = ",str,": StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
  }
/*  Resultado
str = : StringBufferLen(str) = 200   StringLen(str) = 0
*/
```

Véase también

StringBufferLen, StringLen

StringLen

Devuelve el número de símbolos de una cadena.

```
int  StringLen(
   string  string_value      // cadena
   );
```

Parámetros

string_value

[in] Cadena para calcular la longitud.

Valor devuelto

Número de símbolos en la cadena sin contar el cero de terminación.

StringReplace

Reemplaza todas las subcadenas encontradas en una cadena de caracteres por una secuencia de símbolos especificada.

```
int  StringReplace(
   string&         str,              // cadena en la que se realiza el reemplazamient
   const string    find,             // subcadena que se busca
   const string    replacement       // subcadena que será insertada en las posicione
   );
```

Parámetros

str

[in][out] Cadena en la que hay que realizar reemplazos.

find

[in] Subcadena que se busca para ser reemplazada.

replacement

[in] Subcadena que será insertada en lugar de la subcadena encontrada.

Valor devuelto

Número de reemplazos realizados en caso de éxito, de lo contrario devuelve -1. Para obtener el código del error, se debe llamara ala función GetLastError().

Nota

Si la función se ha ejecutado con éxito pero no ha sido realizado ningún reemplazamiento (subcadena a reemplazar no encontrada), la función devuelve 0.

El error puede estar en los parámetros *str* o *find* incorrectos (cadena vacía o no inicializada, más detalles en StringInit()). Además, el error puede surgir si no hay suficiente memoria para completar los reemplazamientos.

Ejemplo:

```
  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);

//  Resultado
//  Replaced: 3. Result=The slow black bear jumped over the lazy dog.
//
```

Véase también

StringSetCharacter(), StringSubstr()

StringSetCharacter

Devuelve copia de una cadena con el valor de símbolo modificado en una posición especificada.

```
bool  StringSetCharacter(
   string&   string_var,       // cadena
   int       pos,              // posición
   ushort    character         // símbolo
   );
```

Parámetros

string_var

[in][out] Cadena.

pos

[in] Posición del símbolo en la cadena. Puede ser de 0 a StringLen(text).

character

[in] Código de símbolo Unicode.

Nota

Si el valor pos es menos que la longitud de la cadena y el valor del código de símbolo = 0, la cadena se recorta (pero el tamaño del buffer distribuido para esta cadena se queda sin cambiar). La longitud de la cadena se iguala a pos.

Si el valor de parámetro *pos* es igual al valor de longitud, el símbolo especificado se añade al final de la cadena, de esta manera la longitud se aumente a uno.

Ejemplo:

```
void OnStart()
  {
   string str="0123456789";
   Print("before: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
//--- introducimos el valor cero entre la cadena
   StringSetCharacter(str,6,0);
   Print("  after: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) = ",StringLen(str));
//--- añadimos el símbolo al final de la cadena
   int size=StringLen(str);
   StringSetCharacter(str,size,'+');
   Print("addition: str = ",str,",StringBufferLen(str) = ",
         StringBufferLen(str),"  StringLen(str) =",StringLen(str));
  }
/* Resultado
   before: str = 0123456789 ,StringBufferLen(str) = 0   StringLen(str) = 10
    after: str = 012345 ,StringBufferLen(str) = 16   StringLen(str) = 6
   addition: str = 012345+ ,StringBufferLen(str) = 16   StringLen(str) = 7
*/
```

Véase también

StringBufferLen, StringLen, StringFill, StringInit

StringSplit

Obtiene las subcadenas desde la cadena especificada según el separador establecido y devuelve el número de subcadenas obtenidas.

```
int  StringSplit(
   const string   string_value,       // cadena para buscar subcadenas
   const ushort   separator,          // separador usando el cual se realizará la bús
   string         & result[]          // array pasado por referencia para obtener las
   );
```

Parámetros

string_value

[in] La cadena desde la cual hay que obtener las subcadenas. La cadena en sí no se cambia.

pos

[in] Código del símbolo del separador. Para obtener el código, se puede usar la función StringGetCharacter().

result[]

[out] Un array de cadenas en el que se colocan las subcadenas obtenidas.

Valor devuelto

Número de cadenas obtenidas en el array result[]. Si en la cadena pasada no se ha encontrado el separador, entonces en el array se colocará sólo una cadena fuente.

Si la cadena string_value está vacía o NULL, la función devolverá cero. En caso del error la función devolverá -1. Para obtener el código del error, hay que llamar a la función GetLastError().

Ejemplo:

```
string to_split="vida_es_bella"; // cadena que hay que dividir en subcadenas
   string sep="_";                // separador en forma de un símbolo
   ushort u_sep;                  // código del separador
   string result[];               // array para obtener cadenas
   //--- obtenemos el código del separador
   u_sep=StringGetCharacter(sep,0);
   //--- dividimos la cadena en subcadenas
   int k=StringSplit(to_split,u_sep,result);
   //--- Mostramos el comentario
   PrintFormat("Se ha obtenido cadenas: %d. Se ha utilizado el separador '%s' con el
   //--- ahora visualizaremos todas las cadenas obtenidas
   if(k>0)
     {
      for(int i=0;i<k;i++)
        {
         PrintFormat("result[%d]=%s",i,result[i]);
        }
     }
```

Véase también

StringReplace(), StringSubstr(), StringConcatenate()

StringSubstr

Extrae una subcadena de una cadena de texto que se empieza desde una posición especificada.

```
string  StringSubstr(
   string  string_value,      // cadena
   int     start_pos,         // posición de inicio
   int     length=-1          // longitud de la cadena a extraer
   );
```

Parámetros

string_value

[in] Cadena de la que se extrae una subcadena.

start_pos

[in] Posición inicial de subcadena. Puede ser de 0 a StringLen(text) -1.

length=-1

[in] Longitud de la cadena a extraer. Si el valor del parámetro es igual a -1, o el parámetro no está definido, entonces la subcadena será extraida empezando desde la posición especificada hasta el final de la cadena.

Valor devuelto

Si es posible devuelve una copia de subcadena extraida. De lo contrario se devuelve una cadena vacía.

StringToLower

Transforma todos los símbolos de una cadena indicada en minúsculas en el lugar.

```
bool  StringToLower(
   string&  string_var      // cadena para procesar
   );
```

Parámetros

string_var

[in][out] Cadena.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false. Para obtener el código de error hay que llamar a la función GetLastError().

StringToUpper

Transforma todos los símbolos de una cadena indicada en mayúsculas en el lugar.

```
bool  StringToUpper(
   string&  string_var      // cadena para procesar
   );
```

Parámetros

string_var

[in][out] Cadena.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false. Para obtener el código de error hay que llamar a la función GetLastError().

StringTrimLeft

Borra los símbolos de salto de línea, espacios y símbolos de tabulación desde el inicio de la cadena hasta el primer símbolo significativo. Cadena se modifica en el lugar.

```
int  StringTrimLeft(
   string&  string_var      // cadena para recortar
   );
```

Parámetros

string_var

[in][out] Cadena que será recortada por la izquierda.

Valor devuelto

Devuelve el número de símbolos cortados.

StringTrimRight

Borra los símbolos de salto de línea, espacios y símbolos de tabulación desde el último símbolo significativo hasta el final de la cadena. Cadena se modifica en el lugar.

```
int  StringTrimRight(
   string&  string_var      // cadena para recortar
   );
```

Parámetros

string_var

[in][out] Cadena que será recortada por la derecha.

Valor devuelto

Devuelve el número de símbolos cortados.

Fecha y hora

Es un grupo de funciones que se usan para trabajar con los datos del tipo datetime (el número entero que representa la cantidad de segundos han pasado desde 0 horas del 1 de Enero del año 1970).

Para organizar los contadores y temporizadores de alta resolución, se debe utilizar la función GetTickCount() que ofrece los valores en millisegundos.

Función	Acción
TimeCurrent	Devuelve la última hora conocida del servidor (la hora de llegada de la última cotización) en el formato datetime
TimeTradeServer	Devuelve la hora actual de calculación del servidor comercial
TimeLocal	Devuelve la hora local del ordenador en el formato datetime
TimeGMT	Devuelve la hora GMT en el formato datetime, teniendo en cuenta el cambio de horarios de verano o de invierno, según la hora local del ordenador en el que está funcionando el terminal de cliente
TimeDaylightSavings	Devuelve el indicio de cambio horario verano/ invierno
TimeGMTOffset	Devuelve la diferencia actual entre la hora GMT y hora local del ordenador en segundos, teniendo en cuenta el cambio horario verano/ invierno
TimeToStruct	Convierte un valor del tipo datetime a una variable del tipo de estructura MqlDateTime
StructToTime	Convierte una variable del tipo de estructura MqlDateTime a un valor del tipo datetime

TimeCurrent

Devuelve la última hora conocida del servidor, la hora de llegada de la última cotización para uno de los símbolos seleccionados de la ventana "Estudio de mercado". En el manejador OnTick() dicha función devolverá la hora del tick procesado que ha llegado. En otras ocasiones (por ejemplo, llamada en los manejadores OnInit(), OnDeinit(), OnTimer(), etc.) es la hora de llegada de la última cotización para cualquier símbolo disponible en la ventana "Estudio de mercado", la misma hora que se muestra en el encabezamiento de esta ventana. El valor de la hora se forma en el servidor comercial y no depende de las configuraciones en el ordenador de usuario. Existen 2 variantes de la función.

Llamada sin parámetros

```
datetime  TimeCurrent();
```

Llamada con el parámetro del tipo MqlDateTime

```
datetime  TimeCurrent(
   MqlDateTime&  dt_struct      // variable del tipo de estructura
   );
```

Parámetros

dt_struct

[out] Variable del tipo de estructura MqlDateTime.

Valor devuelto

Valor del tipo datetime

Nota

Si la variable del tipo de estructura MqlDateTime ha sido pasada como un parámetro, entonces ella se rellena de modo correspondiente.

Para organizar los contadores y temporizadores de alta resolución, se debe utilizar la función GetTickCount() que ofrece los valores en millisegundos.

TimeTradeServer

Devuelve la hora actual de calculación del servidor comercial. A diferencia de la función TimeCurrent (), el cálculo del valor de la hora se realiza en el terminal de cliente y depende de las configuraciones de hora en el ordenador de usuario. Existen 2 variantes de la función.

Llamada sin parámetros

```
datetime  TimeTradeServer();
```

Llamada con el parámetro del tipo MqlDateTime

```
datetime  TimeTradeServer(
   MqlDateTime&  dt_struct      // variable del tipo de estructura
   );
```

Parámetros

dt_struct

[out] Variable del tipo de estructura MqlDateTime.

Valor devuelto

Valor del tipo datetime

Nota

Si la variable del tipo de estructura MqlDateTime ha sido pasada como un parámetro, entonces ella se rellena de modo correspondiente.

Para organizar los contadores y temporizadores de alta resolución, se debe utilizar la función GetTickCount() que ofrece los valores en millisegundos.

TimeLocal

Devuelve la hora local del ordenador en el que está funcionando el terminal de cliente. Existen 2 variantes de la función.

Llamada sin parámetros

```
datetime  TimeLocal();
```

Llamada con el parámetro del tipo MqlDateTime

```
datetime  TimeLocal(
   MqlDateTime&  dt_struct      // variable del tipo de estructura
   );
```

Parámetros

dt_struct

[out] Variable del tipo de estructura MqlDateTime.

Valor devuelto

Valor del tipo datetime

Nota

Si la variable del tipo de estructura MqlDateTime ha sido pasada como un parámetro, entonces ella se rellena de modo correspondiente.

Para organizar los contadores y temporizadores de alta resolución, se debe utilizar la función GetTickCount() que ofrece los valores en millisegundos.

TimeGMT

Devuelve la hora GMT que se calcula, teniendo en cuenta el cambio de horarios de verano o de invierno, según la hora local del ordenador en el que está funcionando el terminal de cliente. Existen 2 variantes de la función.

Llamada sin parámetros

```
datetime  TimeGMT();
```

Llamada con el parámetro del tipo MqlDateTime

```
datetime  TimeGMT(
   MqlDateTime&  dt_struct      // variable del tipo de estructura
   );
```

Parámetros

dt_struct

[out] Variable del tipo de estructura MqlDateTime.

Valor devuelto

Valor del tipo datetime

Nota

Si la variable del tipo de estructura MqlDateTime ha sido pasada como un parámetro, entonces ella se rellena de modo correspondiente.

Para organizar los contadores y temporizadores de alta resolución, se debe utilizar la función GetTickCount() que ofrece los valores en millisegundos.

TimeDaylightSavings

Devuelve la corrección del horario de verano en segundos, si el cambio al horario de verano ha sido realizado. Depende de la configuración de la hora en el ordenador de usuario.

```
int  TimeDaylightSavings();
```

Valor devuelto

Si ha sido realizado el cambio al horario de invierno (estándar), se devuelve 0.

TimeGMTOffset

Devuelve la diferencia actual entre la hora GMT y hora local del ordenador en segundos, teniendo en cuenta el cambio horario verano/invierno. Depende de la configuración de la hora en el ordenador de usuario.

```
int  TimeGMTOffset();
```

Valor devuelto

Valor del tipo int que representa la diferencia actual entre la hora local del ordenador y la hora GMT en segundos.

TimeToStruct

Convierte un valor del tipo datetime (cantidad de segundos transcurridos desde 01.01.1970) a una variable del tipo de estructura MqlDateTime.

```
void  TimeToStruct(
   datetime      dt,            // fecha y hora
   MqlDateTime&  dt_struct      // estructura para recibir valores
   );
```

Parámetros

dt

[in] Valor de la fecha para la conversión.

dt_struct

[out] Variable de estructura del tipo MqlDateTime.

Valor devuelto

No hay valor devuelto.

StructToTime

Convierte una variable del tipo de estructura MqlDateTime a un valor del tipo datetime y devuelve el valor final.

```
datetime  StructToTime(
   MqlDateTime&  dt_struct      // estructura de fecha y hora
   );
```

Parámetros

dt_struct

[in] Variable de estructura del tipo MqlDateTime.

Valor devuelto

Valor del tipo datetime que contiene la cantidad de segundos transcurridos desde 01.01.1970.

Información de cuenta

Las funciones que devuelven los parámetros de la cuenta corriente.

Función	Acción
AccountInfoDouble	Devuelve valor del tipo double de la propiedad correspondiente de la cuenta
AccountInfoInteger	Devuelve valor del tipo de números enteros (bool,int o long) de la propiedad correspondiente de la cuenta
AccountInfoString	Devuelve valor del tipo string de la propiedad correspondiente de la cuenta

AccountInfoDouble

Devuelve el valor de la propiedad correspondiente de la cuenta.

```
double  AccountInfoDouble(
   int  property_id      // identificador de la propiedad
   );
```

Parámetros

property_id

[in] Identificador de la propiedad. El valor puede ser uno de los valores ENUM_ACCOUNT_INFO_DOUBLE.

Valor devuelto

Valor del tipo double.

Ejemplo:

```
void OnStart()
  {
//--- sacamos toda la información que se pueda de la función AccountInfoDouble()
   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_FREEMARGIN =  %G",AccountInfoDouble(ACCOUNT_FREEMARGIN));
   printf("ACCOUNT_MARGIN_LEVEL =  %G",AccountInfoDouble(ACCOUNT_MARGIN_LEVEL));
   printf("ACCOUNT_MARGIN_SO_CALL = %G",AccountInfoDouble(ACCOUNT_MARGIN_SO_CALL));
   printf("ACCOUNT_MARGIN_SO_SO = %G",AccountInfoDouble(ACCOUNT_MARGIN_SO_SO));
  }
```

Véase también

SymbolInfoDouble, SymbolInfoString, SymbolInfoInteger, PrintFormat

AccountInfoInteger

Devuelve el valor de la propiedad correspondiente de la cuenta.

```
long  AccountInfoInteger(
   int  property_id      // identificador de la propiedad
   );
```

Parámetros

property_id

[in] Identificador de la propiedad. El valor puede ser uno de los valores ENUM_ACCOUNT_INFO_INTEGER.

Valor devuelto

Valor del tipo long.

Nota

La propiedad debe ser uno de los tipos bool, int o long.

Ejemplo:

```
void OnStart()
  {
//--- sacamos toda la información que se pueda de la función AccountInfoInteger()
   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)AccountInfoInteger(ACCOU
   ENUM_ACCOUNT_STOPOUT_MODE stopOutMode=(ENUM_ACCOUNT_STOPOUT_MODE)AccountInfoIntege

//--- avisamos sobre la posibilidad de realizar las operaciones comerciales
   if(thisAccountTradeAllowed)
      Print("La actividad comercial para esta cuenta está permitida");
   else
      Print("La actividad comercial para esta cuenta está prohibida!");

//--- comprobamos si los expertos pueden comerciar en esta cuenta
   if(EATradeAllowed)
      Print("La actividad comercial usando los Asesores Expertos está permitida para
   else
      Print("La actividad comercial usando los Asesores Expertos está prohibida para

//--- averiguamos el tipo de cuenta
   switch(tradeMode)
     {
      case(ACCOUNT_TRADE_MODE_DEMO):
         Print("Es una cuenta de demostración");
         break;
```

```
      case(ACCOUNT_TRADE_MODE_CONTEST):
         Print("Es una cuenta de concurso");
         break;
      default:Print("Es una cuenta real!");
     }

//--- averiguamos el régimen de establecimiento del nivel StopOut
   switch(stopOutMode)
     {
      case(ACCOUNT_STOPOUT_MODE_PERCENT):
         Print("El nivel StopOut se especifica en porcentaje");
         break;
      default:Print("El nivel StopOut se especifica en moneda");
     }
  }
```

Véase también

Información de cuenta

AccountInfoString

Devuelve el valor de la propiedad correspondiente de la cuenta.

```
string  AccountInfoString(
   int  property_id      // identificador de la propiedad
   );
```

Parámetros

property_id

[in] Identificador de la propiedad. El valor puede ser uno de los valores ENUM_ACCOUNT_INFO_STRING.

Valor devuelto

Valor del tipo string.

Ejemplo:

```
void OnStart()
  {
//--- sacamos toda la información que se pueda de la función AccountInfoString()
   Print("Nombre del agente = ",AccountInfoString(ACCOUNT_COMPANY));
   Print("Divisa del depósito = ",AccountInfoString(ACCOUNT_CURRENCY));
   Print("Nombre del cliente = ",AccountInfoString(ACCOUNT_NAME));
   Print("Nombre del servidor comercial = ",AccountInfoString(ACCOUNT_SERVER));
  }
```

Véase también

Información de cuenta

Comprobación de estado

Funciones que devuelven los parámetros del estado actual del terminal de cliente

Función	Acción
GetLastError	Devuelve el valor del último error
IsStopped	Devuelve true, si se ha recibido el comando de terminar la ejecución de un programa mql5
UninitializeReason	Devuelve el código de la causa de deinicialización
TerminalInfoInteger	Devuelve un valor del tipo entero de una propiedad correspondiente del ambiente de programa mql5
TerminalInfoString	Devuelve un valor del tipo string de una propiedad correspondiente del ambiente de programa mql5
MQL5InfoInteger	Devuelve un valor del tipo entero de una propiedad correspondiente de un programa mql5 en funcionamiento
MQL5InfoString	Devuelve un valor del tipo string de una propiedad correspondiente de un programa mql5 en funcionamiento
Symbol	Devuelve el nombre de un símbolo del gráfico corriente
Period	Devuelve el período de tiempo del gráfico corriente
Digits	Devuelve la cantidad de dígitos decimales después del punto que determina la precisión de medición del precio del símbolo del gráfico corriente
Point	Devuelve el tamaño del punto del instrumento actual en divisa de cotización

GetLastError

Devuelve el contenido de la variable de sistema _LastError.

```
int  GetLastError();
```

Valor devuelto

Devuelve el valor del último error ocurrido durante la ejecución de un programa mql5.

Nota

Después de llamar a la función el contenido de la variable _LastError no se pone a cero. Para poner esta variable a cero, hay que llamar a la función ResetLastError()

IsStopped

Comprueba el cierre forzoso de un programa mql5.

```
bool  IsStopped();
```

Valor devuelto

Devuelve true, si la variable de sistema _StopFlag contiene un valor distinto a 0. El valor no nulo se escribe en la variable _StopFlag, si ha llegado el comando de terminar la ejecución de un programa mql5. En este caso hay que terminar la ejecución cuanto antes, de lo contrario el programa será cerrado forzosamente desde el exterior dentro de 3 segundos.

UninitializeReason

Devuelve el código de la causa de deinicialización.

```
int  UninitializeReason();
```

Valor devuelto

Devuelve el valor de la variable _UninitReason que se forma antes de la llamada a la función OnDeinit(). Este valor depende del motivo que ha provocado la deinicialización.

TerminalInfoInteger

Devuelve el valor de una propiedad correspondiente del ambiente de programa mql5.

```
int  TerminalInfoInteger(
   int  property_id      // identificador de propiedad
   );
```

Parámetros

property_id

[in] Identificador de una propiedad. Puede ser uno de los valores de la enumeración ENUM_TERMINAL_INFO_INTEGER.

Valor devuelto

Valor del tipo int.

TerminalInfoString

La función devuelve el valor de una propiedad correspondiente del ambiente de programa mql5. La propiedad tiene que ser del tipo string

```
string  TerminalInfoString(
   int  property_id      // identificador de propiedad
   );
```

Parámetros

property_id

[in] Identificador de una propiedad. Puede ser uno de los valores de la enumeración ENUM_TERMINAL_INFO_STRING.

Valor devuelto

Valor del tipo string.

MQL5InfoInteger

Devuelve el valor de una propiedad correspondiente de un programa mql5 en funcionamiento.

```
int  MQL5InfoInteger(
   int  property_id      // identificador de la propiedad
   );
```

Parámetros

property_id

[in] Identificador de la propiedad. Puede ser uno de los valores de la enumeración ENUM_MQL5_INFO_INTEGER.

Valor devuelto

Valor del tipo int.

MQL5InfoString

Devuelve el valor de una propiedad correspondiente de un programa mql5 en funcionamiento.

```
string  MQL5InfoString(
   int  property_id      // identificador de la propiedad
   );
```

Parámetros

property_id

[in] Identificador de la propiedad. Puede ser uno de los valores de la enumeración ENUM_MQL5_INFO_STRING.

Valor devuelto

Valor del tipo string.

Symbol

Devuelve el nombre de un símbolo del gráfico corriente.

```
string  Symbol();
```

Valor devuelto

Valor de la variable de sistema _Symbol, donde se almacena el nombre de un símbolo del gráfico corriente.

Period

Devuelve el período de tiempo del gráfico corriente.

```
ENUM_TIMEFRAMES  Period();
```

Valor devuelto

Contenido de la variable _Period en la que se almacena el valor del período de tiempo del gráfico corriente. Puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES.

Véase también

PeriodSeconds, Períodos de gráficos, Fecha y hora, Visibilidad de objetos

Digits

Devuelve la cantidad de dígitos decimales después del punto que determina la precisión de medición del precio del símbolo del gráfico corriente.

```
int  Digits();
```

Valor devuelto

Valor de la variable _Digits, en la que se almacena el número de dígitos decimales después del punto que determina la precisión del precio del símbolo del gráfico corriente.

Point

Devuelve el tamaño del punto del símbolo corriente en divisa de cotización.

```
double  Point();
```

Valor devuelto

Valor de la variable _Point que almacena el tamaño del punto del símbolo corriente en divisa de cotización.

Obtención de información de mercado

Son las funciones que sirven para conseguir la información sobre el estado del mercado.

Función	Acción
SymbolsTotal	Devuelve la cantidad de símbolos disponibles (seleccionados en MarketWatch o todos)
SymbolName	Devuelve el nombre del símbolo especificado
SymbolSelect	Elige un símbolo o la ventana MarketWatch o remueve un símbolo de la ventana
SymbolInfoDouble	Devuelve valor double del símbolo especificado para la propiedad correspondiente
SymbolInfoInteger	Devuelve valor de un tipo entero (long, datetime, int o bool) del símbolo especificado para la propiedad correspondiente
SymbolInfoString	Devuelve valor string del símbolo especificado para la propiedad correspondiente
SymbolInfoTick	Devuelve precios corrientes para un símbolo especificado en una variable del tipo MqlTick
SymbolInfoSessionQuote	Permite obtener fecha y hora de apertura y de cierre de la especificada sesión de cotización para el símbolo y el día de la semana especificados.
SymbolInfoSessionTrade	Permite obtener fecha y hora de apertura y de cierre de la especificada sesión de compraventa para el símbolo y el día de la semana especificados.
MarketBookAdd	Proporciona la apertura de profundidad de mercado (Depth Market) para un símbolo indicado, además se encarga de la suscripción a las notificaciones acerca del cambio de esta profundidad de mercado
MarketBookRelease	Proporciona el cierre de profundidad de mercado (Depth Market) para un símbolo indicado, además da de baja la suscripción a las notificaciones acerca del cambio de esta profundidad de mercado
MarketBookGet	Devuelve un array de estructuras del tipo MqlBookInfo que contiene datos de la profundidad de mercado de un símbolo especificado

SymbolsTotal

Devuelve la cantidad de símbolos disponibles (seleccionados en MarketWatch o todos).

```
int  SymbolsTotal(
   bool  selected      // true - sólo los símbolos en MarketWatch
   );
```

Parámetros

selected

[in] Modo de solicitud. Puede adquirir valores true o false.

Valor devuelto

Si el parámetro "selected" es igual a true, se devuelve la cantidad de símbolos seleccionados en MarketWatch. Si el valor es false, se devuelve la cantidad total de todos los símbolos.

SymbolName

Devuelve el nombre del símbolo especificado.

```
string  SymbolName(
   int   pos,          // número en la lista
   bool  selected      // true - sólo los símbolos en MarketWatch
   );
```

Parámetros

pos

[in] Número del símbolo según la orden.

selected

[in] Modo de solicitud. Si el valor es true, el símbolo se coge de la lista de símbolos seleccionados en MarketWatch. Si el valor es false, el símbolo se coge de la lista general.

Valor devuelto

Valor del tipo string con el nombre del símbolo.

SymbolSelect

Elige un símbolo o la ventana MarketWatch o remueve un símbolo de la ventana.

```
bool  SymbolSelect(
   string  name,        // nombre del símbolo
   bool    select       // añadir o remover
   );
```

Parámetros

name

[in] Nombre del símbolo.

select

[in] Conmutador. Si el valor es false, este símbolo tiene que ser removido de la ventana MarketWatch, de lo contrario el símbolo tiene que ser seleccionado en la ventana MarketWatch. El símbolo no puede ser removido si hay gráficos abiertos con este símbolo, o hay posiciones abiertas para este símbolo.

Valor devuelto

En caso de fallo la función devuelve false.

SymbolIsSynchronized

Comprueba si los datos para en símbolo especificado en el terminal están sincronizados con los datos en el servidor comercial.

```
bool  SymbolIsSynchronized(
   string  name,        // nombre del símbolo
   );
```

Parámetros

name

[in] Nombre del símbolo.

Valor devuelto

Si los datos están sincronizados, devuelve true, de lo contrario devuelve false.

Véase también

SymbolInfoInteger, Organización de acceso a los datos

SymbolInfoDouble

Devuelve la propiedad correspondiente del símbolo especificado. Existen 2 variantes de la función.

1. Inmediatamente devuelve el valor de la propiedad.

```
double  SymbolInfoDouble(
   string                    name,        // símbolo
   ENUM_SYMBOL_INFO_DOUBLE   prop_id      // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso de éxito el valor de la propiedad se coloca en una variable receptora que se pasa por referencia por el último parámetro.

```
bool  SymbolInfoDouble(
   string                    name,        // símbolo
   ENUM_SYMBOL_INFO_DOUBLE   prop_id,     // identificador de la propiedad
   double&                   double_var   // aquí recibimos el valor de la propiedad
   );
```

Parámetros

name

[in] Nombre del símbolo.

prop_id

[in] Identificador de la propiedad del símbolo. El valor puede ser uno de los valores de la enumeración ENUM_SYMBOL_INFO_DOUBLE.

double_var

[out] Variable del tipo double que recibe el valor de la propiedad solicitada.

Valor devuelto

Valor del tipo double. En caso de la ejecución fallida, la información sobre el error se puede obtener a través de la función GetLastError():

- 5040 - parámetro string erróneo para especificar el nombre del símbolo,
- 4301 - símbolo desconocido (instrumento financiero),
- 4302 - símbolo no seleccionado en "Observación del Mercado" (no figura en la lista de símbolos disponibles),
- 4303 - identificador erróneo de la propiedad del símbolo.

Nota

Si la función se utiliza para recibir la información sobre el último tick, será mejor utilizar SymbolInfoTick(). Es muy posible que para este símbolo aún no haya habido ninguna cotización desde el momento de conexión del terminal a la cuenta de trading. En este caso el valor que se solicita va a ser indeterminado.

En la mayoría de los casos será suficiente utilizar la función SymbolInfoTick() que en una llamada permite conseguir los valores Ask, Bid, Last, Volume y el tiempo de llegada del último tick.

Ejemplo:

```
void OnTick()
  {
//--- obtenemos el spread de las propiedades del símbolo
   bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
   string comm=StringFormat("Spread %s = %I64d puntos\r\n",
                            spreadfloat?"flotante":"fijo",
                            SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- ahora nosotros mismos calculamos el spread
   double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   double spread=ask-bid;
   int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
   comm=comm+"Spread calculado = "+(string)spread_points+" puntos";
   Comment(comm);
  }
```

SymbolInfoInteger

Devuelve la propiedad correspondiente del símbolo especificado. Existen 2 variantes de la función.

1. Inmediatamente devuelve el valor de la propiedad.

```
long  SymbolInfoInteger(
   string                    name,      // símbolo
   ENUM_SYMBOL_INFO_INTEGER  prop_id    // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso de éxito el valor de la propiedad se coloca en una variable receptora que se pasa por referencia por el último parámetro.

```
bool  SymbolInfoInteger(
   string                    name,      // símbolo
   ENUM_SYMBOL_INFO_INTEGER  prop_id,   // identificador de la propiedad
   long&                     long_var   // aquí recibimos el valor de la propiedad
   );
```

Parámetros

name

[in] Nombre del símbolo.

prop_id

[in] Identificador de la propiedad del símbolo. Su valor puede ser uno de los valores de la enumeración ENUM_SYMBOL_INFO_INTEGER.

long_var

[out] Variable del tipo long que recibe el valor de la propiedad solicitada.

Valor devuelto

Valor del tipo long. En caso de la ejecución fallida, la información sobre el error se puede obtener a través de la función GetLastError():

- 5040 - parámetro string erróneo para especificar el nombre del símbolo,
- 4301 - símbolo desconocido (instrumento financiero),
- 4302 - símbolo no seleccionado en "Observación del Mercado" (no figura en la lista de símbolos disponibles),
- 4303 - identificador erróneo de la propiedad del símbolo.

Nota

Si la función se utiliza para recibir la información sobre el último tick, será mejor utilizar SymbolInfoTick(). Es muy posible que para este símbolo aún no haya habido ninguna cotización desde el momento de conexión del terminal a la cuenta de trading. En este caso el valor que se solicita va a ser indeterminado.

En la mayoría de los casos será suficiente utilizar la función SymbolInfoTick() que en una llamada permite conseguir los valores Ask, Bid, Last, Volume y el tiempo de llegada del último tick.

Ejemplo:

```
void OnTick()
  {
//--- obtenemos el spread de las propiedades del símbolo
   bool spreadfloat=SymbolInfoInteger(Symbol(),SYMBOL_SPREAD_FLOAT);
   string comm=StringFormat("Spread %s = %I64d puntos\r\n",
                            spreadfloat?"flotante":"fijo",
                            SymbolInfoInteger(Symbol(),SYMBOL_SPREAD));
//--- ahora nosotros mismos calculamos el spread
   double ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
   double bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   double spread=ask-bid;
   int spread_points=(int)MathRound(spread/SymbolInfoDouble(Symbol(),SYMBOL_POINT));
   comm=comm+"Spread calculado = "+(string)spread_points+" puntos";
   Comment(comm);
  }
```

SymbolInfoString

Devuelve la propiedad correspondiente del símbolo especificado. Existen 2 variantes de la función.

1. Inmediatamente devuelve el valor de la propiedad.

```
string  SymbolInfoString(
   string                    name,        // símbolo
   ENUM_SYMBOL_INFO_STRING   prop_id      // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso de éxito el valor de la propiedad se coloca en una variable receptora que se pasa por referencia por el último parámetro.

```
bool  SymbolInfoString(
   string                    name,        // símbolo
   ENUM_SYMBOL_INFO_STRING   prop_id,     // identificador de la propiedad
   string&                   string_var   // aquí recibimos el valor de la propiedad
   );
```

Parámetros

name

[in] Nombre del símbolo.

prop_id

[in] Identificador de la propiedad del símbolo. Su valor puede ser uno de los valores de la enumeración ENUM_SYMBOL_INFO_STRING.

string_var

[out] Variable del tipo string que recibe el valor de la propiedad solicitada.

Valor devuelto

Valor del tipo string. En caso de la ejecución fallida, la información sobre el error se puede obtener a través de la función GetLastError():

- 5040 - parámetro string erróneo para especificar el nombre del símbolo,
- 4301 - símbolo desconocido (instrumento financiero),
- 4302 - símbolo no seleccionado en "Observación del Mercado" (no figura en la lista de símbolos disponibles),
- 4303 - identificador erróneo de la propiedad del símbolo.

Nota

Si la función se utiliza para recibir la información sobre el último tick, será mejor utilizar SymbolInfoTick(). Es muy posible que para este símbolo aún no haya habido ninguna cotización desde el momento de conexión del terminal a la cuenta de trading. En este caso el valor que se solicita va a ser indeterminado.

En la mayoría de los casos será suficiente utilizar la función SymbolInfoTick() que en una llamada permite conseguir los valores Ask, Bid, Last, Volume y el tiempo de llegada del último tick.

SymbolInfoTick

Devuelve precios corrientes para un símbolo especificado en una variable del tipo MqlTick.

```
bool  SymbolInfoTick(
   string    symbol,      // símbolo
   MqlTick&  tick         // referencia a una estructura
   );
```

Parámetros

symbol

[in] Nombre del símbolo.

tick

[out] Referencia a una estructura del tipo MqlTick, en la que serán colocados los precios corrientes y la hora de la última renovación de precios.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

SymbolInfoSessionQuote

Permite obtener fecha y hora de apertura y de cierre de la especificada sesión de cotización para el símbolo y el día de la semana especificados.

```
bool  SymbolInfoSessionQuote(
   string            name,              // nombre del símbolo
   ENUM_DAY_OF_WEEK  day_of_week,       // día de la semana
   uint              session_index,     // número de sesión
   datetime&         from,              // hora de apertura de la sesión
   datetime&         to                 // hora de cierre de la sesión
   );
```

Parámetros

name

[in] Nombre del símbolo.

ENUM_DAY_OF_WEEK

[in] Día de la semana, se coge el valor de la enumeración ENUM_DAY_OF_WEEK.

uint

[in] Número ordinal de la sesión para la que hay que recibir la hora y fecha de apertura y de cierre. La indexación de las sesiones se empieza desde 0.

from

[out] Hora de apertura de la sesión en segundos transcurridos desde las 00 horas y 00 minutos. En el valor obtenido hay que ignorar la fecha.

to

[out] Hora de cierre de la sesión en segundos transcurridos desde las 00 horas y 00 minutos. En el valor obtenido hay que ignorar la fecha.

Valor devuelto

Si se obtienen los datos respecto a la sesión, símbolo y el día de la semana especificado, entonces la función devuelve true. De lo contrario, se devuelve false.

Véase también

Información sobre el instrumento, TimeToStruct, Estructura de fecha

SymbolInfoSessionTrade

Permite obtener fecha y hora de apertura y de cierre de la especificada sesión de compraventa para el símbolo y el día de la semana especificados.

```
bool  SymbolInfoSessionTrade(
   string            name,              // nombre del símbolo
   ENUM_DAY_OF_WEEK  day_of_week,       // día de la semana
   uint              session_index,     // número de sesión
   datetime&         from,              // hora de apertura de la sesión
   datetime&         to                 // hora de cierre de la sesión
   );
```

Parámetros

name

[in] Nombre del símbolo.

ENUM_DAY_OF_WEEK

[in] Día de la semana, se coge el valor de la enumeración ENUM_DAY_OF_WEEK.

uint

[in] Número ordinal de la sesión para la que hay que recibir la hora y fecha de apertura y de cierre. La indexación de las sesiones se empieza desde 0.

from

[out] Hora de apertura de la sesión en segundos transcurridos desde las 00 horas y 00 minutos. En el valor obtenido hay que ignorar la fecha.

to

[out] Hora de cierre de la sesión en segundos transcurridos desde las 00 horas y 00 minutos. En el valor obtenido hay que ignorar la fecha.

Valor devuelto

Si se obtienen los datos respecto a la sesión, símbolo y el día de la semana, entonces la función devuelve true. De lo contrario, se devuelve false.

Véase también

Información sobre el instrumento, TimeToStruct, Estructura de fecha

MarketBookAdd

Proporciona la apertura de profundidad de mercado (Depth Market) para un símbolo indicado, además se encarga de la suscripción a las notificaciones acerca del cambio de esta profundidad de mercado.

```
bool  MarketBookAdd(
   string  symbol      // símbolo
   );
```

Parámetros

symbol

[in] Nombre del símbolo cuya profundidad de mercado va a ser usada en el Asesor Experto o script.

Valor devuelto

Devuelve valor true en caso de apertura con éxito, de lo contrario devuelve false.

Nota

Normalmente, esta función debe invocarse desde la función OnInit() o en el constructor de clase. Para procesar las notificaciones que llegan en el programa de Asesor Experto tiene que haber la función void OnBookEvent(string& symbol).

Véase también

Estructura de profundidad de mercado, Estructuras y clases

MarketBookRelease

Proporciona el cierre de profundidad de mercado (Depth Market) para un símbolo indicado, además da de baja la suscripción a las notificaciones acerca del cambio de esta profundidad de mercado.

```
bool  MarketBookRelease(
   string  symbol      // nombre del símbolo
   );
```

Parámetros

symbol

[in] Nombre del símbolo.

Valor devuelto

Devuelve valor true en caso de cierre con éxito, de lo contrario devuelve false.

Nota

Normalmente, esta función debe invocarse desde la función OnDeinit(), si la función correspondiente MarketBookAdd() ha sido invocada en la función OnInit(). O tiene que llamarse del destructor de clase, si la función correspondiente MarketBookAdd() se invoca en el constructor de esta clase.

Véase también

Estructura de profundidad de mercado, Estructuras y clases

MarketBookGet

Devuelve un array de estructuras del tipo MqlBookInfo que contiene datos de la profundidad de mercado de un símbolo especificado.

```
bool  MarketBookGet(
   string         symbol,      // símbolo
   MqlBookInfo&   book[]       // referencia a un array
   );
```

Parámetros

symbol

[in] Nombre del símbolo.

book[]

[out] Referencia a un array del historial de profundidad de mercado. El array puede ser previamente distribuido para una cantidad suficiente de apuntes. Si un array dinámico no ha sido previamente distribuido en la memoria operativa,el terminal de cliente lo distribuirá por sí mismo.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

La profundidad de mercado debe ser abierta previamente por la función MarketBookAdd().

Ejemplo:

```
   MqlBookInfo priceArray[];
   bool getBook=MarketBookGet(NULL,priceArray);
   if(getBook)
     {
      int size=ArraySize(priceArray);
      Print("MarketBookInfo para ",Symbol());
      for(int i=0;i<size;i++)
        {
         Print(i+":",priceArray[i].price
               +"    Volume = "+priceArray[i].volume,
               " type = ",priceArray[i].type);
        }
     }
   else
     {
      Print("No se ha podido obtener el contenido de la profundidad de mercado para e
     }
```

Véase también

Estructura de profundidad de mercado, Estructuras y clases

Acceso a las series temporales y a los datos de indicadores

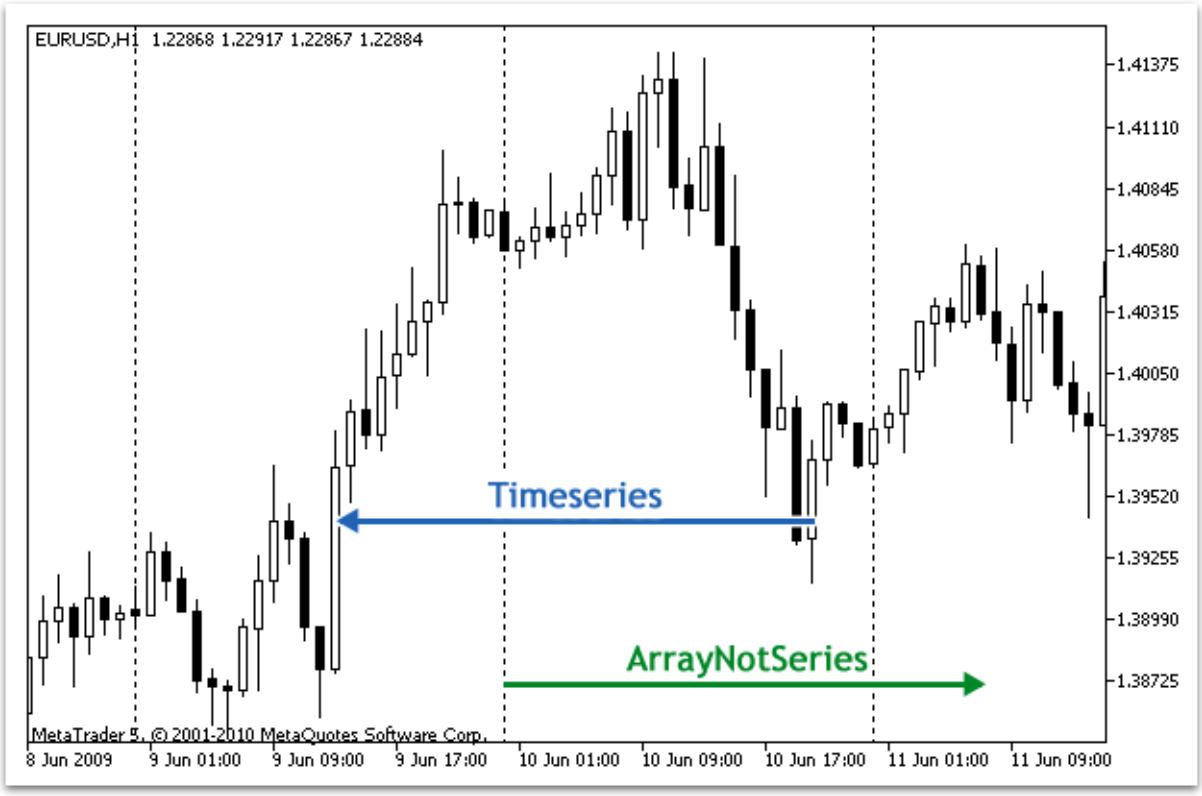
Estas son las funciones para trabajar con las series temporales e indicadores. Una serie temporal se diferencia de una matriz usual en que la indexación de los elementos de una serie temporal se realiza del final de la matriz al principio (de los datos más recientes a los más antiguos). Se recomienda usar sólo los arrays dinámicos para copiar los valores de series temporales e indicadores, porque las funciones de copiado están diseñadas para asignar de una manera independiente el tamaño necesario de los arrays que reciben los valores.

Hay una **importante excepción** de esta regla: si necesitamos copiar las series temporales y valores de indicadores con mucha frecuencia, por ejemplo, con cada nueva llamada a OnTick() en los Asesores Expertos o con cada nueva invocación de OnCalculate() en los indicadores, entonces sería mejor usar los arrays distribuidos estáticamente porque **las operaciones de asignación de memoria** para las matrices dinámicas **requieren su tiempo adicional** y eso tendrá su efecto durante los procesos de prueba y optimización.

Cuando usamos las funciones de acceso a las series temporales y a los valores de indicadores, hay que tener en cuenta la dirección de la indexación. Esto está descrito con detalles en la sección llamada Dirección de indexación en los arrays y series temporales.

El acceso a los datos de los indicadores y series temporales se realiza independientemente del hecho de disposición de estos datos solicitados (llamado el acceso asíncrono). Es sumamente importante para la calculación de los indicadores personalizados, por eso si los datos solicitados no están disponibles, las funciones como *Copy...()* inmediatamente devuelven el error. Sin embargo, si accedemos desde los Asesores Expertos o scripts, se hacen varios intentos de obtener los datos con una pequeña pausa que se necesita para proporcionar el tiempo necesario para cargar las series temporales que faltan o para calcular los valores de indicadores.

En el apartado Organización de acceso a los datos se explican minuciosamente los detalles de la obtención, almacenamiento y solicitud de los datos de precios en el terminal de cliente MetaTrader 5.



Históricamente se ha constituido que el acceso a los datos en un array de precios se realiza desde el final de los datos. Físicamente los datos nuevos siempre se añaden al final del array, pero el índice de este array siempre es igual a cero. El índice 0 en una matriz-serie temporal significa los datos de la barra en curso, es decir, de la barra que corresponde al intervalo de tiempo no finalizado en dicho período de tiempo (timeframe).

Un período de tiempo (timeframe) es un plazo de tiempo durante el cual se forma una barra de precio. En total están predefinidos 21 períodos de tiempo estándares.

Función	Acción
SeriesInfoInteger	Devuelve la información sobre el estado de datos históricos
Bars	Devuelve la cantidad de barras en el historial por símbolo y período correspondientes
BarsCalculated	Devuelve la cantidad de datos calculados en el búfer de indicadores o -1 en caso del error (los datos aún no están calculados)
IndicatorCreate	Devuelve el manejador (handle) del indicador técnico especificado que ha sido creado a base del array de parámetros del tipo MqlParam
IndicatorParameters	Devuelve para el manejador especificado el número de los parámetros de entrada del indicador, así como los propios valores y el tipo de parámetros

IndicatorRelease	Elimina el manejador (handle) del indicador y libera la parte calculadora del indicador si nadie la está usando
CopyBuffer	Recibe en el array los datos de un búfer especificado desde un indicador especificado
CopyRates	Recibe en un array los datos históricos de la estructura Rates para un símbolo y período especificados
CopyTime	Recibe en un array los datos históricos sobre el tiempo de apertura de barras para un símbolo y período especificados
CopyOpen	Recibe en un array los datos históricos sobre el precio de apertura de barras para un símbolo y período especificados
CopyHigh	Recibe en un array los datos históricos sobre el precio máximo de barras para un símbolo y período especificados
CopyLow	Recibe en un array los datos históricos sobre el precio mínimo de barras para un símbolo y período especificados
CopyClose	Recibe en un array los datos históricos sobre el precio de cierre de barras para un símbolo y período especificados
CopyTickVolume	Recibe en un array los datos históricos sobre volúmenes de tick para un símbolo y período especificados
CopyRealVolume	Recibe en un array los datos históricos sobre volúmenes comerciales para un símbolo y período especificados
CopySpread	Recibe en un array los datos históricos sobre los spreads para un símbolo y período especificados

A pesar de que mediante la función ArraySetAsSeries() para los arrays se pueda establecer un modo de acceso a los elementos igual que para las series temporales, hay que recordar que físicamente los elementos de un array siempre se almacenan en el mismo orden, sólo se cambia la dirección de la indexación. Para demostrarlo vamos a realizar el siguiente ejemplo:

```
   datetime TimeAsSeries[];
//--- Establecemos el acceso al array como a una serie temporal
   ArraySetAsSeries(TimeAsSeries,true);
   ResetLastError();
   int copied=CopyTime(NULL,0,0,10,TimeAsSeries);
```

```
   if(copied<=0)
     {
      Print("No se ha podido copiar la hora de apertura de las últimas 10 barras");
      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("No se ha podido copiar la hora de apertura de las últimas 10 barras");
      return;
     }
   size=ArraySize(ArrayNotSeries);
   for(int i=size-1;i>=0;i--)
     {
      Print("ArrayNotSeries["+i+"] = ",ArrayNotSeries[i]);
     }
```

Por lo tanto se mostrará algo parecido a lo siguiente:

```
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
```

Como vemos, para el array TimeAsSeries con el aumento del índice el valor de tiempo con este índice se disminuye, es decir, nos movemos del presente al pasado. Para un array estándar ArrayNotSeries ocurre todo lo contrario; con el aumento del índice nos movemos del pasado al presente.

Véase también

ArrayIsDynamic, ArrayGetAsSeries, ArraySetAsSeries, ArrayIsSeries

Dirección de indexación en los arrays y series temporales

Por defecto todas los arrays y búfers de indicadores tienen la dirección de indexación de izquierda a derecha. El índice del primer elemento siempre es igual a cero. De esta manera, el primer elemento de un array o búfer de indicador con el índice 0 por defecto se encuentra en el extremo izquierdo y el último elemento se encuentra en el extremo derecho.

Un búfer de indicador es un array dinámico del tipo double, cuyo tamaño es gestionado por el terminal de cliente para que éste siempre corresponda a la cantidad de barras sobre las cuales se calcula el indicador. Un array dinámico habitual del tipo double se asigna como un búfer de indicador a través de la función SetIndexBuffer(). Para los buffers de indicadores no hace falta establecer el tamaño mediante la función ArrayResize(), esto será hecho por el sistema de ejecución del mismo terminal.

Las series temporales son arrays con la indexación inversa. Es decir, el primer elemento de una serie temporal se encuentra en el extremo derecho y el último se encuentra en el izquierdo. Las series temporales están destinadas para almacenar los datos históricos de precios de los instrumentos financieros y contienen obligatoriamente la información sobre la hora, entonces se puede decir que en una serie temporal los datos más recientes se encuentran en el extremo derecho y los más antiguos en el extremo izquierdo.

Por tanto, en una serie temporal el elemento con el índice cero contiene la información sobre la última cotización de un instrumento. Si una serie temporal representa datos del período de tiempo diario, en la posición cero se contienen datos sobre el día en curso no finalizado, y en la posición con el índice uno se almacenan los datos del día de ayer.

Cambio de dirección de indexación

La función ArraySetAsSeries() permite cambiar el modo de acceso a los elementos de un array dinámico, pero el orden físico de almacenamiento de datos en la memoria del ordenador no sufre cambio alguno. Esta función simplemente cambia el modo de direccionamiento hacia los elementos de un array, por eso cuando copiamos un array dentro del otro a través de la función ArrayCopy(), el contenido del array-receptor no va a depender de la dirección de indexación en el array fuente.

No se puede cambiar la dirección de indexación para los arrays distribuidos estáticamente. Incluso si un array ha sido pasado como un parámetro a una función, dentro de esta función los intentos de cambiar la dirección de indexación no tendrán efecto alguno.

Para los búfers de indicadores, igual que para los arrays habituales, también se permite establecer la dirección de indexación al revés, como en las series temporales. Es decir, en este caso la referencia hacia la posición cero en el búfer de indicador va a significar la referencia hacia el último valor en el búfer de indicador correspondiente, y esto va a corresponder al valor del indicador en la última barra. Pero como ya se ha mencionado antes, la ubicación física de datos en el búfer de indicador se queda sin cambiar.

Recepción de datos de precios en los indicadores

Cada indicador personalizado obligatoriamente ha de contener la función OnCalculate(). A esta función se le pasan los datos de precios necesarios para calcular los valores en los búfers de indicadores. A través de la función ArrayGetAsSeries() se puede averiguar la dirección de indexación en estos arrays pasados.

Los arrays pasados a la función reflejan los datos de precios, es decir, estos arrays tienen los indicios de las series temporales y la función ArrayIsSeries() devolverá true a la hora de comprobar estos arrays. Pero a pesar de eso, en cualquier caso hay que comprobar la dirección de indexación sólo a través de la función ArrayGetAsSeries().

Para no depender de los valores por defecto, hay que llamar incondicionalmente a la función ArraySetAsSeries() para los arrays con los que se prevé trabajar y establecer la dirección de indexación necesaria.

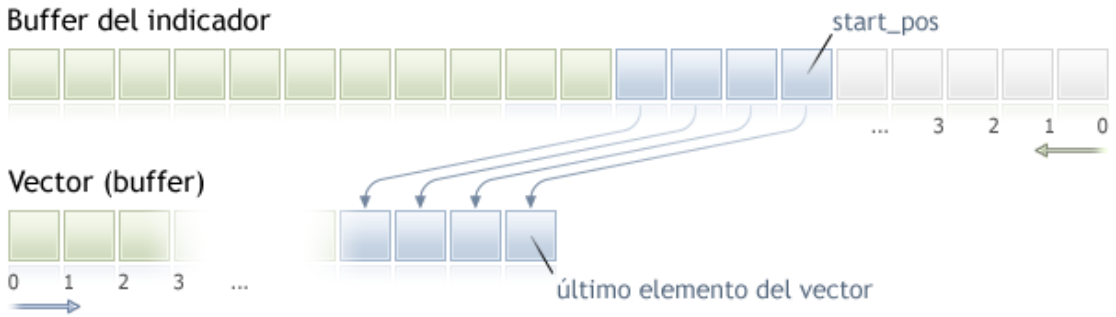
Recepción de datos de precios y valores de indicadores

Por defecto en los Asesores Expertos, indicadores y scripts todos los arrays tienen la dirección de indexación de izquierda a derecha. En caso de necesidad, en cualquier programa mql5 se puede solicitar los valores de las series temporales referente a cualquier símbolo y período de tiempo, además, los valores de indicadores calculados sobre cualquier símbolo y período de tiempo.

Para conseguir estos datos se usan las funciones Copy...():

- CopyBuffer - copia los valores de un búfer de indicador a un array del tipo double;
- CopyRates - copia el historial de precios a una matriz de estructuras MqlRates;
- CopyTime - copia los valores Time a un array del tipo datetime;
- CopyOpen - copia los valores Open a un array del tipo double;
- CopyHigh - copia los valores High a un array del tipo double;
- CopyLow - copia los valores Low a un array del tipo double;
- CopyClose - copia los valores Close a un array del tipo double;
- CopyTickVolume - copia los volúmenes de tick a un array del tipo long;
- CopyRealVolume - copia los volúmenes bursátiles a un array del tipo long;
- CopySpread - copia el historial de spreads a un array del tipo int;

Todas estas funciones trabajan de una manera igual, y por eso será suficiente estudiar el mecanismo de obtención de datos en el ejemplo de CopyBuffer(). Se supone que todos los datos solicitados tienen la dirección de indexación igual que en las series temporales, y en la posición con el índice 0 (cero) se almacenan los datos de la barra actual no finalizada. Para conseguir acceder a estos datos necesitamos copiar el volumen de datos que hace falta a un array-receptor, por ejemplo al array *buffer*.



Durante el copiado es necesario indicar la posición de inicio en el array fuente a partir de la cual los

datos empiezan a copiarse al array de destino. En caso del éxito la cantidad de elementos especificada será copiada desde el array de origen (en este caso se trata del búfer de indicador) al array-receptor. El copiado siempre se realiza tal como se muestra en el dibujo, independientemente de la dirección de indexación establecida en el array-receptor.

Si se supone procesar los datos de precios en un ciclo con el gran número de iteraciones, entonces se recomienda comprobar el hecho de la finalización forzosa del programa utilizando la función IsStopped ():

```
int copied=CopyBuffer(ma_handle,// manejador del indicador
                      0,        // índice del búfer de indicador
                      0,        // posición de inicio para el copiado
                      number,   // número de valores a copiar
                      Buffer    // array-receptor de valores
                      );
if(copied<0) return;
int k=0;
while(k<copied && !IsStopped())
  {
   //--- obtenemos el valor para el índice k
   double value=Buffer[k];
   // ...
   // trabajo con el valor value
   k++;
  }
```

Ejemplo:

```
input int per=10; // período del exponente
int ma_handle;    // manejador del indicador
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   ma_handle=iMA(_Symbol,0,per,0,MODE_EMA,PRICE_CLOSE);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---
   double ema[10];
   int copied=CopyBuffer(ma_handle,// manejador del indicador
                         0,        // índice del búfer de indicador
                         0,        // posición de inicio para el copiado
```

```
                           10,       // número de valores a copiar
                           ema       // array-receptor de valores
                           );
   if(copied<0) return;
// .... código que sigue
  }
```

Véase también

Organización de acceso a los datos

Organización de acceso a los datos

En esta sección vamos a estudiar las cuestiones relacionadas con la obtención, almacenamiento y solicitud de datos de precios (series temporales).

Recepción de datos del servidor comercial

Antes de que los datos de precios estén disponibles en el terminal MetaTrader 5, es necesario obtener y procesarlos. Para recibir los datos hay que conectarse al terminal comercial MetaTrader 5. Los datos llegan del servidor a petición del terminal en forma de unos bloques de barras de un minuto bien empaquetados.

El mecanismo de dirigirse al servidor para obtener los datos no depende del modo de presentar la solicitud, sea por el usuario navegando por el gráfico o sea de modo de programación en el lenguaje MQL5.

Almacenamiento de datos intermedios

Los datos recibidos del servidor se despaquetan automáticamente y se guardan en el formato intermedio especial HCC. Los datos de cada símbolo se colocan en una carpeta individual *directorio_de_terminal*\bases*nombre_de_servidor*\history*nombre_de_símbolo*. Por ejemplo, los datos del símbolo EURUSD recibidos del servidor comercial MetaQuotes-Demo estarán guardados en la carpeta *directorio_de_terminal*\bases\MetaQuotes-Demo\history\EURUSD\.

Los datos se escriben en los archivos con la extensión .hcc, cada archivo almacena datos de barras de minutos de un año. Por ejemplo, el archivo 2009.hcc de la carpeta EURUSD contiene datos de minutos del símbolo EURUSD del año 2009. Estos archivos se usan para la preparación de datos de precios de todos los períodos y no están destinados para el acceso directo.

Obtención de datos de un período necesario desde los datos intermedios

Los archivos auxiliares en el formato HCC desempeñan el papel de la fuente de información a la hora de construir los datos de precios para los períodos de tiempo solicitados en el formato HC. Los datos en el formato HC son series temporales que están preparadas al máximo para el acceso rápido. Se crean únicamente a la petición de un gráfico o un programa mql5 en el volumen que no supera el valor del parámetro "Max bars in charts" y se guardan en los archivos con la extensión hc para su posterior uso.

Para ahorrar los recursos, los datos relacionados con un período de tiempo se cargan y se almacenan en la memoria operativa sólo si es preciso, en caso si éstos no son requeridos durante un plazo de tiempo considerable se realiza su descarga de la memoria operativa y ellos se guardan en el archivo. Para cada período de tiempo los datos se preparan independientemente de la presencia de datos ya preparados para otros períodos. Las reglas de formación y accesibilidad de datos son iguales para todos los períodos de tiempo. Es decir, a pesar de que la unidad de almacenamiento de datos en el formato HCC sea una barra de un minuto, la disponibilidad de datos en el formato HCC no significa la disponibilidad y accesibilidad de datos del período M1 con el formato HC en el mismo volumen.

La recepción de nuevos datos desde el servidor provoca la renovación automática de datos de precios utilizados en el formato HC de todos los períodos. Esto también lleva al recálculo de todos los indicadores que los usan implícitamente como datos de entrada para el cálculo.

Parámetro "Max bars in chart"

El parámetro "Max bars in charts" limita la cantidad de barras en el formato HC disponible para los gráficos, indicadores y programas mql5. Esta limitación concierne a los datos de todos los períodos de tiempo, y sirve principalmente para ahorrar recursos del ordenador.

Al establecer los valores altos del dicho parámetro, tenemos que recordar que tratándose de un historial bastante profundo de datos de precios para los períodos temporales menores, el consumo de memoria para almacenar series temporales y buffers de indicadores puede ser de centenares de megabytes, llegando al límite de memoria operativa para el terminal de cliente (2GB para las aplicaciones MS Windows de 32 bits).

Los cambios del parámetro "Max bars in charts" tendrán su efecto una vez reiniciado el terminal de cliente. De por sí el cambio de dicho parámetro no implica ni la llamada automática al servidor a por los datos adicionales, ni la formación de barras adicionales de una serie temporal. La solicitud de datos de precios adicionales y la renovación de series temporales se hacen en caso de desplazar el gráfico en la zona de datos que faltan, o bien, solicitando los datos que faltan desde un programa mql5.

El volumen de datos solicitados al servidor corresponde a la cantidad requerida de barras de dicho período, teniendo en cuenta el valor del parámetro "Max bars in charts". La limitación puesta por el parámetro no es tan rigurosa, y en algunas ocasiones el número de barras disponibles en el período temporal puede superar insignificadamente el valor del parámetro corriente.

Disponibilidad de datos

La presencia de datos en el formato HCC, o incluso en el formato HC listo para utilizarse, no siempre supone la disponibilidad absoluta de estos datos para ser mostrados en un gráfico o para utilizarlos en programas mql5.

A la hora de acceder a los datos de precios o a los valores de indicadores desde los programas mql5, hay que recordar que su disponibilidad en un momento dado o desde un momento dado no está garantizada. Esto está relacionado con lo siguiente: con el fin de ahorrar los recursos en MetaTrader 5, no se almacena una copia completa de datos requeridos para un programa mql5, sino se proporciona un acceso directo a la base de datos del terminal.

El historial de precios para todos los períodos temporales se forma de datos comunes en el formato HCC y cualquier renovación de datos desde el servidor supone la renovación de datos para todos los períodos temporales y recálculo de indicadores. A consecuencia de esto, el acceso a los datos puede estar bloqueado, incluso si éstos estaban disponibles hace un momento.

Sincronización de datos del terminal y datos del servidor

Puesto que un programa mql5 puede dirigirse a los datos por cualquier símbolo y período de tiempo, cabe posibilidad que los datos de la serie temporal requerida todavía no están formados en el terminal o los datos de precios requeridos no están sincronizados con el servidor comercial. En este caso es muy complicado pronosticar el tiempo de espera.

Los algoritmos que usan los ciclos de latencia no es la mejor solución. La única excepción en este caso son los scripts, puesto que ellos no tienen otra elección de algoritmo debido a no disponer del procesamiento de eventos. Para los indicadores personalizados dichos algoritmos, así como otros ciclos de latencia, no se recomiendan en absoluto porque llevan a parar el cálculo de todos los

indicadores y otro procesamiento de datos de precios para dicho símbolo.

Para los Asesores Expertos e indicadores personalizados es mejor usar el modelo de eventos de procesamiento. Si durante el manejo del evento OnTick() o OnCalculate() no hemos llegado a obtener todos los datos necesarios de la serie temporal requerida, hay que salir del manejador de eventos, esperando que, al invocar el manejador la próxima vez, obtengamos el acceso a los datos.

Ejemplo de un script para descargar el historial

Vamos a ver un ejemplo - un script ejecuta la solicitud de recibir el historial acerca de un instrumento especificado desde el servidor comercial. Este script sirve para inicializar el instrumento necesario en el gráfico, el período de tiempo no importa, puesto que, y como se decía antes, los datos de precios llegan del servidor en forma de unos datos de minutos empaquetados, de los cuales, luego, se construye una serie temporal predeterminada.

Vamos a organizar todas las acciones de recepción de datos a través de una función independiente CheckLoadHistory(symbol, timeframe, start_date):

```
int CheckLoadHistory(string symbol,ENUM_TIMEFRAMES period,datetime start_date)
  {
  }
```

La función CheckLoadHistory() está pensada como una función universal a la que se puede llamar desde un programa cualquiera (Asesor Experto, script o indicador), por tanto hace falta tres parámetros de entrada: nombre del símbolo, período y fecha de inicio a partir de la cual necesitamos el historial de precios.

Vamos a insertar en el código de la función todas las comprobaciones necesarias antes de solicitar el historial que nos falta. Antes de toda hay que asegurarse que el nombre del símbolo y valor del período son correctos:

```
   if(symbol==NULL || symbol=="") symbol=Symbol();
   if(period==PERIOD_CURRENT)     period=Period();
```

En el siguiente paso nos persuadimos de que el símbolo especificado esté disponible en la ventana MarketWatch, es decir, el historial para este símbolo va a estar disponible cuando se presenta la solicitud al servidor comercial. Si éste no se encuentra en dicha ventana, hay que añadirlo usando la función SymbolSelect().

```
   if(!SymbolInfoInteger(symbol,SYMBOL_SELECT))
     {
      if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
      SymbolSelect(symbol,true);
     }
```

Ahora hace falta recibir la fecha de inicio del historial ya disponible para el par símbolo/período especificado. Es posible que el valor del parámetro de entrada startdate, pasado a la función CheckLoadHistory(), entre en el intervalo del historial ya disponible, entonces no hace falta presentar ninguna solicitud al servidor comercial. En este momento la función SeriesInfoInteger() con el modificador SERIES_FIRSTDATE sirve para obtener la primera fecha para el símbolo/período.

```
   SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
   if(first_date>0 && first_date<=start_date) return(1);
```

Otra verificación importante es la comprobación del tipo de programa desde el cual la función es invocada. Acordemos que el envío de la solicitud para actualizar la serie temporal con el mismo período que tiene el indicador que llama esta actualización es muy indeseable. Esta indeseabilidad está condicionada con el hecho de que la actualización de datos históricos se realice en el mismo hilo en el que opera el indicador. Por eso la posibilidad de clinch es alta. Para la comprobación vamos a usar la función MQL5InfoInteger() con el modificador MQL5_PROGRAM_TYPE.

```
   if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Sy
      return(-4);
```

Si hemos pasado todas las comprobaciones con éxito, vamos a hacer el último intento de evitar acudir al servidor comercial. Primero averiguaremos la fecha de inicio para la que estén disponibles los datos de minuto en el formato HCC. Vamos a solicitar este valor usando la función SeriesInfoInteger() con el modificador SERIES_TERMINAL_FIRSTDATE, y volvemos a compararlo con el valor del parámetro start_date.

```
   if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
     {
      //--- there is loaded data to build timeseries
      if(first_date>0)
        {
         //--- force timeseries build
         CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
         //--- check date
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
        }
     }
```

Si después de todas las comprobaciones el hilo de ejecución sigue estando en el cuerpo de la función CheckLoadHistory(), esto quiere decir que hay necesidad de solicitar los datos de precios que faltan al servidor comercial. Para empezar averiguaremos el valor "Max bars in chart" usando la función TerminalInfoInteger():

```
  int max_bars=TerminalInfoInteger(TERMINAL_MAXBARS);
```

Lo vamos a necesitar para no solicitar los datos de más. Luego averiguaremos la primera fecha en el historial del símbolo en el servidor comercial (sin tener en cuenta el período) mediante la función ya conocida SeriesInfoInteger() con el modificador SERIES_SERVER_FIRSTDATE.

```
   datetime first_server_date=0;
   while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_dat
      Sleep(5);
```

Puesto que la solicitud es una operación asincrónica, la función se invoca en el ciclo con un pequeño retraso de 5 milisegundos hasta que la variable first_server_date adquiera un valor, o la ejecución del ciclo sea interrumpida por el usuario (IsStopped(), en este caso devuelve el valor true). Vamos a indicar un valor correcto de la fecha de inicio, desde la cual empezamos a solicitar los datos de precios en el servidor comercial.

```
   if(first_server_date>start_date) start_date=first_server_date;
   if(first_date>0 && first_date<first_server_date)
      Print("Warning: first server date ",first_server_date,
            " for ",symbol," does not match to first series date ",first_date);
```

Si de pronto la fecha de inicio *first_server_date* del servidor resulta ser menos que la fecha de inicio

first_date del símbolo en el formato HCC, en el diario de registro aparecerá el aviso correspondiente.

Ahora estamos preparados a solicitar los datos de precios que nos faltan al servidor comercial. Vamos a presentar la solicitud en forma del ciclo y empezaremos a rellenar su cuerpo:

```
   while(!IsStopped())
     {
      //1. esperar la sincronización entre la serie temporal reconstruida y el histor
      //2. recibir el número corriente de barras en esta serie temporal
      //   si la cantidad de barras supera el valor de Max_bars_in_chart, podemos sal
      //3. obtenemos la fecha de inicio first_date en la serie temporal reconstruida
      //   start_date si first_date es menos que start_date, podemos salir, el trabaj
      //4. Solicitamos al servidor comercial nueva parte del historial de 100 barras
      //   barra disponible numerada "bars"
     }
```

Los tres primeros puntos se implementan por medios ya conocidos.

```
   while(!IsStopped())
     {
      //--- 1.esperamos que se termine el proceso de reconstrucción de la serie tempo
      while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
         Sleep(5);
      //--- 2.preguntamos cuántas barras tenemos disponibles
      int bars=Bars(symbol,period);
      if(bars>0)
        {
         //--- el número de baras es superior a lo que podemos mostrar en el gráfico,
         if(bars>=max_bars) return(-2);
         //--- 3. averiguamos la fecha de inicio corriente en la serie temporal
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            // la fecha de inicio es más temprana que la solicitada, tarea cumplida
            if(first_date>0 && first_date<=start_date) return(0);
        }
      //4. Solicitamos al servidor comercial nueva parte del historial de 100 barras
      //   empezando desde la última barra disponible numerada "bars"
     }
```

Nos queda el último cuarto punto, que es la misma solicitud del historial. No podemos dirigirnos al servidor directamente pero cualquier función-Copy automáticamente inicia el envío de tal solicitud del terminal al servidor comercial si el historial en el formato HCC no es suficiente. Ya que el tiempo de la primera fecha de inicio en la variable *first_date* es el criterio más fácil y natural para evaluar el grado de ejecución de la solicitud, lo más fácil será usar la función CopyTime().

Cuando llamamos a las funciones que se encargan de copiar cualquier dato desde las series temporales, hay que tener en cuenta que el parámetro *start* (número de la barra a partir de la cual se inicia el copiado de datos de precios) siempre tiene que estar dentro de los límites del historial del terminal disponible. Si disponemos sólo de 100 barras, no tiene sentido intentar copiar 300 barras empezando de la barra con el índice 500. Tal solicitud se entenderá como errónea y no será procesada, es decir, no se cargará ningún historial desde el servidor comercial.

Precisamente por eso vamos a copiar 100 barras de una vez, empezando de la barra con el índice bars. Esto proporciona la carga fluida del historial desde el servidor comercial. En realidad se cargará un poco más de 100 barras solicitadas, es que el servidor envía el historial con una cantidad de información ligeramente sobrepasada.

```
   int copied=CopyTime(symbol,period,bars,100,times);
```

Después de la operación de copiado hay que analizar el número de elementos copiados. Si el intento ha sido fallido, el valor de la variable *copied* será igual a cero y el valor del contador fail_cnt será aumentado a 1. El trabajo de la función será detenido después de 100 intentos fallidos.

```
int fail_cnt=0;
...
   int copied=CopyTime(symbol,period,bars,100,times);
   if(copied>0)
     {
      //--- comprobamos los datos
      if(times[0]<=start_date)  return(0);  // el valor copiado es menos, listo
      if(bars+copied>=max_bars) return(-2); // hay más barras que cabe en el gráfico,
      fail_cnt=0;
     }
   else
     {
      //--- no más de 100 intentos fallidos seguidos
      fail_cnt++;
      if(fail_cnt>=100) return(-5);
      Sleep(10);
     }
```

De esta manera, en la función no sólo está implementado el correcto procesamiento de la situación corriente en cada momento de ejecución, sino también se devuelve el código de finalización, el que podemos manejar después de invocar la función CheckLoadHistory() con el fin de obtener información adicional. Por ejemplo, de esta manera:

```
   int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
   switch(res)
     {
      case -1 : Print("Símbolo desconocido ",InpLoadedSymbol);
      case -2 : Print("Cantidad superior de barras solicitadas a la que puede ser mos
      case -3 : Print("Ejecución interrumpida por el usuario");
      case -4 : Print("Indicador no debe cargar sus propios datos");
      case -5 : Print("Carga fallida");
      case  0 : Print("Todos los datos están cargados");
      case  1 : Print("Cantidad de datos ya disponibles en la serie temporal es sufic
      case  2 : Print("Serie temporal está construida con los datos disponibles en el
      default : Print("Resultado de ejecución no determinado");
     }
```

El código entero de la función viene en el ejemplo del script que demuestra el modo correcto de organizar el acceso a cualquier dato con el procesamiento del resultado de solicitud.

Código:

```
//+------------------------------------------------------------------+
//|                                                TestLoadHistory.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.02"
#property script_show_inputs
//--- input parameters
input string          InpLoadedSymbol="NZDUSD";   // Symbol to be load
input ENUM_TIMEFRAMES InpLoadedPeriod=PERIOD_H1;  // Period to be load
input datetime        InpStartDate=D'2006.01.01'; // Start date
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   Print("Start load",InpLoadedSymbol+","+GetPeriodName(InpLoadedPeriod),"from",InpSt
//---
   int res=CheckLoadHistory(InpLoadedSymbol,InpLoadedPeriod,InpStartDate);
   switch(res)
     {
      case -1 : Print("Unknown symbol ",InpLoadedSymbol);              break;
      case -2 : Print("Requested bars more than max bars in chart"); break;
      case -3 : Print("Program was stopped");                         break;
      case -4 : Print("Indicator shouldn't load its own data");       break;
      case -5 : Print("Load failed");                                 break;
      case  0 : Print("Loaded OK");                                   break;
      case  1 : Print("Loaded previously");                           break;
      case  2 : Print("Loaded previously and built");                 break;
      default : Print("Unknown result");
     }
//---
   datetime first_date;
   SeriesInfoInteger(InpLoadedSymbol,InpLoadedPeriod,SERIES_FIRSTDATE,first_date);
   int bars=Bars(InpLoadedSymbol,InpLoadedPeriod);
   Print("First date",first_date,"-",bars,"bars");
//---
  }
//+------------------------------------------------------------------+
//|                                                                  |
//+------------------------------------------------------------------+
int CheckLoadHistory(string symbol,ENUM_TIMEFRAMES period,datetime start_date)
  {
   datetime first_date=0;
   datetime times[100];
//--- check symbol & period
   if(symbol==NULL || symbol=="") symbol=Symbol();
   if(period==PERIOD_CURRENT)     period=Period();
//--- check if symbol is selected in the MarketWatch
   if(!SymbolInfoInteger(symbol,SYMBOL_SELECT))
     {
      if(GetLastError()==ERR_MARKET_UNKNOWN_SYMBOL) return(-1);
      SymbolSelect(symbol,true);
     }
//--- check if data is present
   SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date);
   if(first_date>0 && first_date<=start_date) return(1);
//--- don't ask for load of its own data if it is an indicator
   if(MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR && Period()==period && Sy
```

```
      return(-4);
//--- second attempt
   if(SeriesInfoInteger(symbol,PERIOD_M1,SERIES_TERMINAL_FIRSTDATE,first_date))
     {
      //--- there is loaded data to build timeseries
      if(first_date>0)
        {
         //--- force timeseries build
         CopyTime(symbol,period,first_date+PeriodSeconds(period),1,times);
         //--- check date
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(2);
        }
     }
//--- max bars in chart from terminal options
   int max_bars=TerminalInfoInteger(TERMINAL_MAXBARS);
//--- load symbol history info
   datetime first_server_date=0;
   while(!SeriesInfoInteger(symbol,PERIOD_M1,SERIES_SERVER_FIRSTDATE,first_server_dat
      Sleep(5);
//--- fix start date for loading
   if(first_server_date>start_date) start_date=first_server_date;
   if(first_date>0 && first_date<first_server_date)
      Print("Warning: first server date ",first_server_date,
```

```
            " for ",symbol," does not match to first series date ",first_date);
//--- load data step by step
   int fail_cnt=0;
   while(!IsStopped())
     {
      //--- wait for timeseries build
      while(!SeriesInfoInteger(symbol,period,SERIES_SYNCHRONIZED) && !IsStopped())
         Sleep(5);
      //--- ask for built bars
      int bars=Bars(symbol,period);
      if(bars>0)
        {
         if(bars>=max_bars) return(-2);
         //--- ask for first date
         if(SeriesInfoInteger(symbol,period,SERIES_FIRSTDATE,first_date))
            if(first_date>0 && first_date<=start_date) return(0);
        }
      //--- copying of next part forces data loading
      int copied=CopyTime(symbol,period,bars,100,times);
      if(copied>0)
        {
         //--- check for data
         if(times[0]<=start_date)  return(0);
         if(bars+copied>=max_bars) return(-2);
         fail_cnt=0;
        }
      else
        {
         //--- no more than 100 failed attempts
         fail_cnt++;
         if(fail_cnt>=100) return(-5);
         Sleep(10);
        }
     }
//--- stopped
   return(-3);
  }
//+------------------------------------------------------------------+
//| devuelve a la cadena valor del período                           |
//+------------------------------------------------------------------+
string GetPeriodName(ENUM_TIMEFRAMES period)
  {
   if(period==PERIOD_CURRENT) period=Period();
//---
   switch(period)
     {
      case PERIOD_M1:  return("M1");
      case PERIOD_M2:  return("M2");
      case PERIOD_M3:  return("M3");
      case PERIOD_M4:  return("M4");
      case PERIOD_M5:  return("M5");
      case PERIOD_M6:  return("M6");
      case PERIOD_M10: return("M10");
      case PERIOD_M12: return("M12");
      case PERIOD_M15: return("M15");
      case PERIOD_M20: return("M20");
      case PERIOD_M30: return("M30");
      case PERIOD_H1:  return("H1");
      case PERIOD_H2:  return("H2");
      case PERIOD_H3:  return("H3");
      case PERIOD_H4:  return("H4");
```

```
      case PERIOD_H6:  return("H6");
      case PERIOD_H8:  return("H8");
      case PERIOD_H12: return("H12");
      case PERIOD_D1:  return("Daily");
      case PERIOD_W1:  return("Weekly");
      case PERIOD_MN1: return("Monthly");
     }
//---
   return("unknown period");
  }
```

SeriesInfoInteger

Devuelve la información sobre el estado de datos históricos. Existen 2 variantes de la función.

Directamente devuelve el valor de la propiedad.

```
long  SeriesInfoInteger(
   string                      symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES             timeframe,        // período
   ENUM_SERIES_INFO_INTEGER    prop_id,          // identificador de la propiedad
   );
```

Devuelve true o false dependiendo del éxito de ejecución de la función.

```
bool  SeriesInfoInteger(
   string                      symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES             timeframe,        // período
   ENUM_SERIES_INFO_INTEGER    prop_id,          // identificador de la propiedad
   long&                       long_var          // variable para recibir la informació
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

prop_id

[in] Identificador de la propiedad solicitada, valor de la enumeración ENUM_SERIES_INFO_INTEGER.

long_var

[out] Variable en la que se coloca el valor de la propiedad solicitada.

Valor devuelto

En primer caso se devuelve el valor del tipo long.

Para el segundo caso, la función devuelve true, si dicha propiedad está disponible y su valor ha sido colocado en la variable long_var, de lo contrario devuelve false. Para obtener información adicional sobre un error, hay que llamar a la función GetLastError().

Ejemplo:

```
void OnStart()
  {
//---
   Print("Cantidad total de barras para el símbolo-período en este momento = ",
         SeriesInfoInteger(Symbol(),0,SERIES_BARS_COUNT));

   Print("La primera fecha para el símbolo-período en este momento = ",
         (datetime)SeriesInfoInteger(Symbol(),0,SERIES_FIRSTDATE));

   Print("La primera fecha en el historial del servidor para el símbolo = ",
         (datetime)SeriesInfoInteger(Symbol(),0,SERIES_SERVER_FIRSTDATE));

   Print("Datos del símbolo están sincronizados = ",
         (bool)SeriesInfoInteger(Symbol(),0,SERIES_SYNCHRONIZED));
  }
```

Bars

Devuelve el número de barras del historial para el símbolo y período correspondientes. Existen 2 variantes de la función.

Solicitar el número de barras en el historial

```
int  Bars(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   );
```

Solicitar el número de barras en el intervalo establecido

```
int  Bars(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   datetime         stop_time         // fecha y hora de finalización
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

Valor devuelto

Si los parámetros start_time y stop_time están especificados, la función devolverá el número de barras en el rango de fechas. Si estos parámetros no están especificados, la función devolverá el número total de barras.

Nota

Si en el momento cuando se llama a la función Bars(), los datos para la serie temporal con los parámetros especificados todavía no están formados en el terminal, o los datos de la serie temporal no están sincronizados con el servidor comercial a la hora de invocar la función, esta función devolverá el valor cero.

Ejemplo:

```
int bars=Bars(_Symbol,_Period);
if(bars>0)
  {
   Print("Cantidad de barras en el historial del terminal para el símbolo-período
  }
else  //no hay barras disponibles
  {
   //--- por lo visto, datos por el símbolo no están sincronizados con los datos d
   bool synchronized=false;
   //--- contador del ciclo
   int attempts=0;
   // hagamos 5 intentos de esperar la sincronización
   while(attempts<5)
     {
      if(SeriesInfoInteger(Symbol(),0,SERIES_SYNCHRONIZED))
        {
         //--- sincronización con éxito, salimos
         synchronized=true;
         break;
        }
      //--- aumentamos el contador
      attempts++;
      //--- esperaremos 10 milisegundos hasta la siguiente iteración
      Sleep(10);
     }
   //--- salimos del ciclo después de sincronización
   if(synchronized)
     {
      Print("Cantidad de barras en el historial del terminal para el símbolo-perío
      Print("La primera fecha en el historial del terminal para el símbolo-período
            (datetime)SeriesInfoInteger(Symbol(),0,SERIES_FIRSTDATE));
      Print("La primera fecha en el historial del servidor para el símbolo = ",
            (datetime)SeriesInfoInteger(Symbol(),0,SERIES_SERVER_FIRSTDATE));
     }
   //--- no se ha conseguido la sincronización de datos
   else
     {
      Print("No se ha podido obtener la cantidad de barras para ",_Symbol);
     }
  }
```

Véase también

Funciones de procesamiento de eventos

BarsCalculated

Devuelve la cantidad de datos calculados para el indicador especificado.

```
int  BarsCalculated(
   int       indicator_handle,     // manejador del indicador
   );
```

Parámetros

indicator_handle

[in] Manejador del indicador obtenido por la función de indicador correspondiente.

Valor devuelto

Devuelve la cantidad de datos calculados en el búfer de indicadores, o devuelve -1 en caso del error (datos aún no han sido calculados).

Nota

Esta función es útil cuando es necesario conseguir los datos de indicador inmediatamente después de su creación (recepción del manejador del indicador).

Ejemplo:

```
void OnStart()
  {
   double Ups[];
//--- establecemos para los arrays el indicio de una serie temporal
   ArraySetAsSeries(Ups,true);
//--- creamos el manejador del indicador Fractals
   int FractalsHandle=iFractals(NULL,0);
//--- reinicializamos el código del error
   ResetLastError();
//--- intentaremos copiar el valor del indicador
   int i,copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
   if(copied<=0)
     {
      Sleep(50);
      for(i=0;i<100;i++)
        {
         if(BarsCalculated(FractalsHandle)>0)
            break;
         Sleep(50);
        }
      copied=CopyBuffer(FractalsHandle,0,0,1000,Ups);
      if(copied<=0)
        {
         Print("Fallo al copiar fractales superiores. Error = ",GetLastError(),
         "i = ",i,"    copied = ",copied);
         return;
        }
```

```
      else
        Print("Los fractales superiores se han copiado con éxito.",
        "i = ",i,"    copied = ",copied);
    }
  else Print("Los fractales superiores se han copiado con éxito. ArraySize = ",Array
}
```

IndicatorCreate

Devuelve el manejador del indicador técnico especificado que ha sido creado a base del array de parámetros del tipo MqlParam.

```
int  IndicatorCreate(
   string           symbol,                    // nombre del símbolo
   ENUM_TIMEFRAMES  period,                    // período
   ENUM_INDICATOR   indicator_type,            // tipo del indicador desde la
   int              parameters_cnt=0,          // número de parámetros
   const MqlParam&  parameters_array[]=NULL,   // array de parámetros
   );
```

Parámetros

symbol

[in] Nombre de símbolo del instrumento, a base de los datos de la cual se calcula el indicador. NULL significa el símbolo actual.

period

[in] Valor del período de tiempo puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el período de tiempo corriente.

indicator_type

[in] Tipo del indicador, puede ser uno de los valores de la enumeración ENUM_INDICATOR.

parameters_cnt

[in] Número de parámetros pasados en el array parameters_array[]. Los elementos del array tienen un tipo especial de la estructura MqlParam. Por defecto, valor cero - los parámetros no se pasan. Si el número de parámetros indicado es distinto a cero, entonces el parámetro *parameters_array* es obligatorio. Se puede pasar no más de 256 parámetros.

parameters_array[]=NULL

[in] Array del tipo MqlParam, cuyos elementos contienen el tipo y valor de cada uno de los parámetros de entrada del indicador técnico.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Si se crea el manejador del indicador del tipo IND_CUSTOM, el campo *type* del primer elemento del array de los parámetros de entrada *parameters_array* obligatoriamente tiene que tener el valor TYPE_STRING desde la enumeración ENUM_DATATYPE, y el campo *string_value* del primer elemento tiene que tener el nombre del indicador personalizado. El indicador personalizado tiene que estar compilado (archivo con la extensión EX5) y ubicarse en el directorio MQL5/Indicators del terminal de cliente o en una subcarpeta.

Los indicadores que requieren verificaciones se definen automáticamente desde la llamada a la función iCustom(), si el parámetro correspondiente es fijado por una cadena constante. Para los demás casos (uso de función IndicatorCreate() o uso de una cadena no constante en el parámetro

que asigna el nombre del indicador) esta propiedad #property tester_indicator es necesaria:

```
#property tester_indicator "indicator_name.ex5"
```

Si en un indicador personalizado se utiliza la primera forma de la llamada, entonces a la hora de pasar los parámetros de entrada, a través del último parámetro se puede especificar adicionalmente a base de qué datos éste va a ser calculado. Si el parámetro "Apply to" no está especificado explícitamente, por defecto el cálculo se basa en los valores PRICE_CLOSE.

Ejemplo:

```
void OnStart()
  {
   MqlParam params[];
   int      h_MA,h_MACD;
//--- create iMA("EURUSD",PERIOD_M15,8,0,MODE_EMA,PRICE_CLOSE);
   ArrayResize(params,4);
//--- set ma_period
   params[0].type         =TYPE_INT;
   params[0].integer_value=8;
//--- set ma_shift
   params[1].type         =TYPE_INT;
   params[1].integer_value=0;
//--- set ma_method
   params[2].type         =TYPE_INT;
   params[2].integer_value=MODE_EMA;
//--- set applied_price
   params[3].type         =TYPE_INT;
   params[3].integer_value=PRICE_CLOSE;
//--- create MA
   h_MA=IndicatorCreate("EURUSD",PERIOD_M15,IND_MA,4,params);
//--- create iMACD("EURUSD",PERIOD_M15,12,26,9,h_MA);
   ArrayResize(params,4);
//--- set fast ma_period
   params[0].type         =TYPE_INT;
   params[0].integer_value=12;
//--- set slow ma_period
   params[1].type         =TYPE_INT;
   params[1].integer_value=26;
//--- set smooth period for difference
   params[2].type         =TYPE_INT;
   params[2].integer_value=9;
//--- set indicator handle as applied_price
   params[3].type         =TYPE_INT;
   params[3].integer_value=h_MA;
//--- create MACD based on moving average
   h_MACD=IndicatorCreate("EURUSD",PERIOD_M15,IND_MACD,4,params);
//--- use indicators
//--- . . .
//--- release indicators (first h_MACD)
   IndicatorRelease(h_MACD);
   IndicatorRelease(h_MA);
  }
```

IndicatorParameters

Devuelve para el manejador especificado el número de los parámetros de entrada del indicador, así como los propios valores y el tipo de parámetros.

```
int  IndicatorParameters(
   int               indicator_handle,     // manejador del indicador
   ENUM_INDICATOR&   indicator_type,       // variable para recibir el tipo del indic
   MqlParam&         parameters[]          // array para recibir parámetros
   );
```

Parámetros

indicator_handle

[in] Manejador del indicador para el que hace falta averiguar el número de parámetros sobre los cuales está calculado éste.

indicator_type

[out] Variable del tipo ENUM_INDICATOR en la que será escrito el tipo del indicador.

parameters[]

[out] Array dinámico para recibir los valores del tipo MqlParam en el que será escrita la lista de parámetros del indicador. El tamaño del array es devuelto por la misma función IndicatorParameters().

Valor devuelto

El número de los parámetros de entrada del indicador con el manejador especificado. En caso del error devuelve -1. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Ejemplo:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {

//--- número de ventanas sobre el gráfico (siempre hay por lo menos una ventana princ
   int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
//--- vamos por las ventanas del gráfico
   for(int w=0;w<windows;w++)
     {
      //--- número de indicadores en esta ventana/subventana
      int total=ChartIndicatorsTotal(0,w);
      //--- repasaremos todos los indicadores en la ventana
      for(int i=0;i<total;i++)
        {
         //--- obtenemos el nombre corto del indicador
         string name=ChartIndicatorName(0,w,i);
         //--- obtenemos el manejador del indicador
```

```
         int handle=ChartIndicatorGet(0,w,name);
         //--- mostramos en el diario
         PrintFormat("Window=%d,  indicator #%d,  handle=%d",w,i,handle);
         //---
         MqlParam parameters[];
         ENUM_INDICATOR indicator_type;
         int params=IndicatorParameters(handle,indicator_type,parameters);
         //--- encabezamiento del mensaje
         string par_info="Short name "+name+", type "
                         +EnumToString(ENUM_INDICATOR(indicator_type))+"\r\n";
         //---
         for(int p=0;p<params;p++)
           {
            par_info+=StringFormat("parameter %d: type=%s, long_value=%d, double_valu
                                   p,
                                   EnumToString((ENUM_DATATYPE)parameters[p].type),
                                   parameters[p].integer_value,
                                   parameters[p].double_value,
                                   parameters[p].string_value
                                   );
           }
         Print(par_info);
        }
      //--- hemos pasado por todos los indicadores en la ventana
     }
//---
  }
```

Véase también

ChartIndicatorGet()

IndicatorRelease

Elimina el manejador del indicador y libera la parte calculadora del indicador si nadie la usa.

```
bool  IndicatorRelease(
   int       indicator_handle     // manejador del indicador
   );
```

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

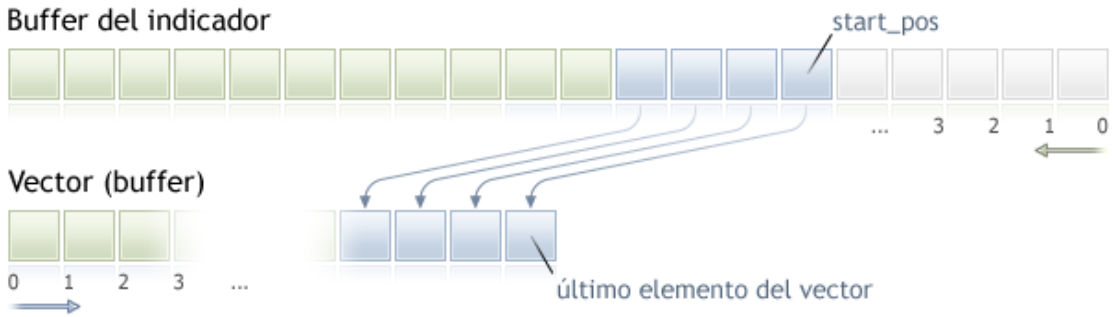
Esta función permite eliminar el manejador del indicador si no se necesita más, y así permite ahorrar el espacio de la memoria. El manejador se elimina inmediatamente, mientras que la parte calculadora del indicador se elimina dentro de un rato (si no hay más llamadas a ésta).

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                        Test_IndicatorRelease.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- input parameters
input int                MA_Period=15;
input int                MA_shift=0;
input ENUM_MA_METHOD     MA_smooth=MODE_SMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
//--- vamos a almacenar el manejador del indicador
int MA_handle;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- creamos el manejador del indicador
   MA_handle=iMA(Symbol(),0,MA_Period,MA_shift,MA_smooth,price);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- si el valor de la variable global todavía no está indicado
   if(GlobalVariableCheck("MA_value")==0)
     {
      //--- array dinámico para los valores del indicador
      double v[];
      //--- obtenemos el valor del indicador en dos últimas barras
      if(CopyBuffer(MA_handle,0,0,2,v)==2 && v[1]!=EMPTY_VALUE)
        {
         //--- recordamos en la variable global valor en la penúltima barra
         if(GlobalVariableSet("MA_value",v[1]))
           {
            //--- liberamos el manejador del indicador
            if(!IndicatorRelease(MA_handle))
               Print("IndicatorRelease() failed. Error ",GetLastError());
           }
        }
     }
//---
  }
```

CopyBuffer

Recibe en el array búfer los datos del búfer especificado del indicador indicado en cantidad especificada.



La cuenta de elementos a copiar (búfer de indicadores con el índice buffer_num) desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente (valor del indicador para la barra corriente).

Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino buffer[], puesto que la función CopyBuffer() trata de ajustar el tamaño del array-receptor en función del espacio que ocupan los datos que se copian. Si un búfer de indicador (array preasignado por la función SetIndexBufer() para almacenar los valores del indicador) se usa como el array-receptor buffer[], entonces se admite el copiado parcial. Podemos ver su ejemplo en el indicador personalizado Awesome_Oscillator.mq5 que entra en el pack estándar del terminal.

Si tenemos que copiar parcialmente los valores del indicador a otro array (que no sea un búfer de indicador), para eso hay que usar un array intermedio donde se copia la cantidad necesaria, y luego de este array-intermediario copiar la cantidad necesaria elemento por elemento a lugares correspondientes del array de destino.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La propiedad del array de destino no importa, sea as_series=true o sea as_series=false. Los datos van a ser copiados de tal manera que el elemento más antiguo estará al principio de la memoria física que ha sido destinada para el array. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopyBuffer(
   int       indicator_handle,     // manejador del indicador
   int       buffer_num,           // número del buffer del indicador
   int       start_pos,            // posición de inicio
   int       count,                // cantidad de datos a copiar
   double    buffer[]              // array de destino en el que se copian los datos
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopyBuffer(
   int       indicator_handle,     // manejador del indicador
   int       buffer_num,           // número del buffer del indicador
   datetime  start_time,           // fecha y hora de inicio
   int       count,                // cantidad de datos a copiar
   double    buffer[]              // array de destino en el que se copian los datos
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopyBuffer(
   int       indicator_handle,     // manejador del indicador
   int       buffer_num,           // número del buffer del indicador
   datetime  start_time,           // fecha y hora de inicio
   datetime  stop_time,            // fecha y hora de finalización
   double    buffer[]              // array de destino en el que se copian los datos
   );
```

Parámetros

indicator_handle

[in] Manejador del indicador recibido por la función de indicador correspondiente.

buffer_num

[in] Número del búfer de indicadores.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

buffer[]

[out] Array del tipo double.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la

serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                TestCopyBuffer3.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot MA
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input bool               AsSeries=true;
input int                period=15;
input ENUM_MA_METHOD     smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
input int                shift=0;
//--- indicator buffers
double                   MABuffer[];
int                      ma_handle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
   Print("Parámetro AsSeries = ",AsSeries);
   Print("Búfer de indicadores después de SetIndexBuffer() es una serie temporal = ",
         ArrayGetAsSeries(MABuffer));
//--- set short indicator name
   IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period+")"+AsSeries);
//--- set AsSeries(dependes from input parameter)
   ArraySetAsSeries(MABuffer,AsSeries);
   Print("Búfer de indicadores después de ArraySetAsSeries(MABuffer,true); es una ser
         ArrayGetAsSeries(MABuffer));
```

```
//---
   ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
   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[])
  {
//--- check if all data calculated
   if(BarsCalculated(ma_handle)<rates_total) return(0);
//--- we can copy not all data
   int to_copy;
   if(prev_calculated>rates_total || prev_calculated<=0) to_copy=rates_total;
   else
     {
      to_copy=rates_total-prev_calculated;
      //--- last value is always copied
      to_copy++;
     }
//--- try to copy
   if(CopyBuffer(ma_handle,0,0,to_copy,MABuffer)<=0) return(0);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

En el ejemplo de arriba se nos muestra que el búfer de indicadores se rellena con valores de otro búfer de indicadores perteneciente a indicador del mismo símbolo/período.

Puede encontrar un ejemplo más detallado de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes estilos de construcción:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,

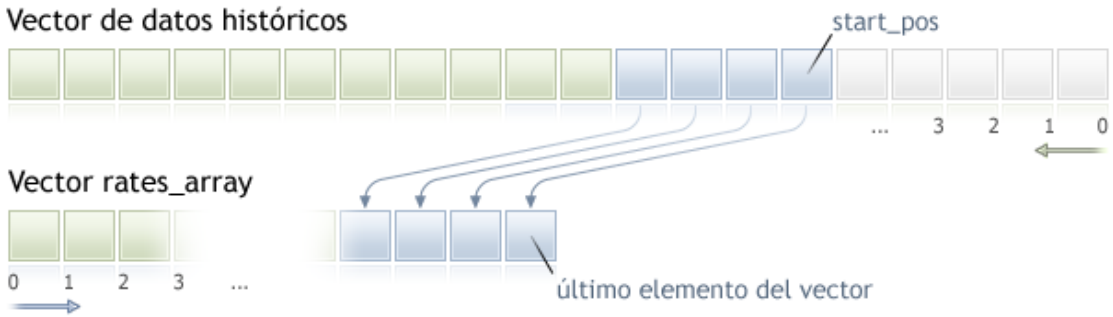
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

Véase también

Propiedades de indicadores personalizados, SetIndexBuffer

CopyRates

Recibe en el array rates_array datos históricos de la estructura MqlRates del símbolo-período especificado en cantidad especificada. La cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente.



Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino, porque si la cantidad de datos resulta ser menos (o más) de la que cabe en el array, entonces se intenta redistribuirlo de tal manera para que los datos solicitados quepan íntegramente.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopyRates(
   string           symbol_name,       // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,         // período
   int              start_pos,         // posición de inicio
   int              count,             // cantidad de datos a copiar
   MqlRates         rates_array[]      // array de destino en el que se copian los da
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopyRates(
   string           symbol_name,       // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,         // período
   datetime         start_time,        // fecha y hora de inicio
   int              count,             // cantidad de datos a copiar
   MqlRates         rates_array[]      // array de destino en el que se copian los da
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopyRates(
```

```
   string           symbol_name,       // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,         // período
   datetime         start_time,        // fecha y hora de inicio
   datetime         stop_time,         // fecha y hora de finalización
   MqlRates         rates_array[]      // array de destino en el que se copian los da
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_time

[in] Hora de la barra correspondiente al primer elemento.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

stop_time

[in] Hora de la barra correspondiente al último elemento.

rates_array[]

[out] Array del tipo MqlRates.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Si el intervalo de datos solicitados se encuentra totalmente fuera del rango de datos disponibles del servidor, la función devuelve -1. Si los datos solicitados salen del rango de TERMINAL_MAXBARS (la cantidad máxima de barras en el gráfico), la función tambien devuelve -1.

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout, pero el historial seguirá cargándose y con la siguiente solicitud del mismo tipo la función devolverá más datos.

Cuando se solicitan los datos por la fecha inicial y el número de elementos requeridos, sólo se devuelven los datos cuya fecha es menor (anterior) o igual a la especificada. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la fecha de apertura de cualquier barra

para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre es igual o menor que la especificada.

Cuando se solicitan los datos de un período de fechas especificado, se devuelven los datos que entran sólo en este intervalo temporal especificado. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la hora de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre se encuentra en el intervalo especificado.

Por ejemplo, si el día corriente es sabado, al intentar copiar los datos del margen semanal indicando *start_time=Último_Martes* y *stop_time=Último_Viernes*, la función devolverá 0 porque la apertura en un período de tiempo semanal siempre cae en domingo, pero ninguna barra semanal no entra en el período especificado.

Si se necesita obtener el valor que corresponde a una barra corriente no finalizada, se puede usar la primera forma de llamada, indicando *start_pos*=0 y *count*=1.

Ejemplo:

```
void OnStart()
  {
//---
   MqlRates rates[];
   ArraySetAsSeries(rates,true);
   int copied=CopyRates(Symbol(),0,0,100,rates);
   if(copied>0)
     {
      Print("Barras copiadas: "+copied);
      string format="open = %G, high = %G, low = %G, close = %G, volume = %d";
      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("Fallo al recibir datos históricos para el símbolo ",Symbol());
  }
```

Puede encontrar un ejemplo más detallado de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes

estilos de construcción:

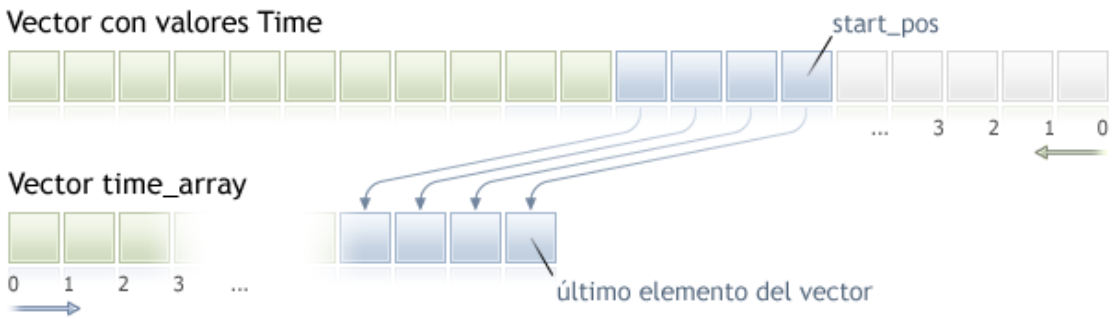
- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

Véase también

Estructuras y clases, TimeToString, StringFormat

CopyTime

La función recibe en el array time_array datos históricos de la hora de apertura de barras para el par especificado de símbolo-período en cantidad especificada. Cabe mencionar que la cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente.



Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino, porque si la cantidad de datos resulta ser menos (o más) de la que cabe en el array, entonces se intenta redistribuirlo de tal manera para que los datos solicitados quepan íntegramente.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La propiedad del array de destino no importa, sea as_series=true o sea as_series=false. Los datos van a ser copiados de tal manera que el elemento más antiguo estará al principio de la memoria física que ha sido destinada para el array. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopyTime(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   int              start_pos,        // posición de inicio
   int              count,            // cantidad de datos a copiar
   datetime         time_array[]      // array para copiar la hora de apertura
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopyTime(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   int              count,            // cantidad de datos a copiar
   datetime         time_array[]      // array para copiar la hora de apertura
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopyTime(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   datetime         stop_time,        // fecha y hora de finalización
   datetime         time_array[]      // array para copiar la hora de apertura
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

time_array[]

[out] Array del tipo datetime.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Si el intervalo de datos solicitados se encuentra totalmente fuera del rango de datos disponibles del servidor, la función devuelve -1. Si los datos solicitados salen del rango de TERMINAL_MAXBARS (la cantidad máxima de barras en el gráfico), la función tambien devuelve -1.

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout, pero el historial seguirá cargándose y con la siguiente solicitud del mismo tipo la función devolverá más datos.

Cuando se solicitan los datos por la fecha inicial y el número de elementos requeridos, sólo se devuelven los datos cuya fecha es menor (anterior) o igual a la especificada. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la fecha de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre es igual o menor que la especificada.

Cuando se solicitan los datos de un período de fechas especificado, se devuelven los datos que entran sólo en este intervalo temporal especificado. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la hora de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre se encuentra en el intervalo especificado.

Por ejemplo, si el día corriente es sabado, al intentar copiar los datos del margen semanal indicando *start_time=Último_Martes* y *stop_time=Último_Viernes*, la función devolverá 0 porque la apertura en un período de tiempo semanal siempre cae en domingo, pero ninguna barra semanal no entra en el período especificado.

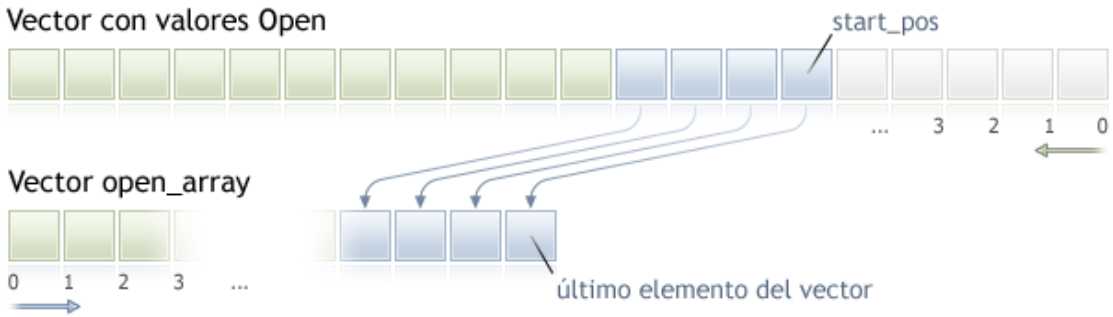
Si se necesita obtener el valor que corresponde a una barra corriente no finalizada, se puede usar la primera forma de llamada, indicando *start_pos*=0 y *count*=1.

Puede encontrar un ejemplo de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes estilos de construcción:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyOpen

La función recibe en el array open_array datos históricos de los precios de apertura de barras para el par especificado de símbolo-período en cantidad especificada. Cabe mencionar que la cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente.



Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino, porque si la cantidad de datos resulta ser menos (o más) de la que cabe en el array, entonces se intenta redistribuirlo de tal manera para que los datos solicitados quepan íntegramente.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La propiedad del array de destino no importa, sea as_series=true o sea as_series=false. Los datos van a ser copiados de tal manera que el elemento más antiguo estará al principio de la memoria física que ha sido destinada para el array. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopyOpen(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   int              start_pos,        // posición de inicio
   int              count,            // cantidad de datos a copiar
   double           open_array[]      // array para copiar los precios de apertura
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopyOpen(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   int              count,            // cantidad de datos a copiar
   double           open_array[]      // array para copiar los precios de apertura
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopyOpen(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   datetime         stop_time,        // fecha y hora de finalización
   double           open_array[]      // array para copiar los precios de apertura
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

open_array[]

[out] Array del tipo double.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Si el intervalo de datos solicitados se encuentra totalmente fuera del rango de datos disponibles del servidor, la función devuelve -1. Si los datos solicitados salen del rango de TERMINAL_MAXBARS (la cantidad máxima de barras en el gráfico), la función tambien devuelve -1.

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout, pero el historial seguirá cargándose y con la siguiente solicitud del mismo tipo la función devolverá más datos.

Cuando se solicitan los datos por la fecha inicial y el número de elementos requeridos, sólo se devuelven los datos cuya fecha es menor (anterior) o igual a la especificada. Este intervalo se

establece y se cuenta con la precisión de segundos. Es decir, la fecha de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre es igual o menor que la especificada.

Cuando se solicitan los datos de un período de fechas especificado, se devuelven los datos que entran sólo en este intervalo temporal especificado. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la hora de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre se encuentra en el intervalo especificado.

Por ejemplo, si el día corriente es sabado, al intentar copiar los datos del margen semanal indicando *start_time=Último_Martes* y *stop_time=Último_Viernes*, la función devolverá 0 porque la apertura en un período de tiempo semanal siempre cae en domingo, pero ninguna barra semanal no entra en el período especificado.

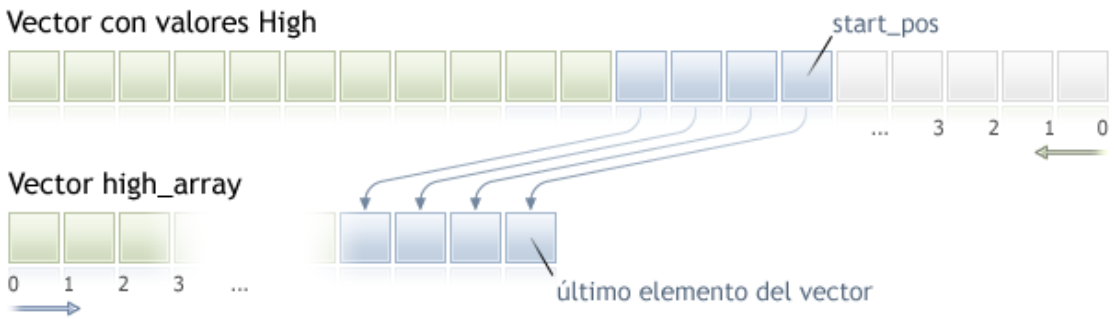
Si se necesita obtener el valor que corresponde a una barra corriente no finalizada, se puede usar la primera forma de llamada, indicando *start_pos*=0 y *count*=1.

Puede encontrar un ejemplo de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes estilos de construcción:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyHigh

La función recibe en el array high_array datos históricos de los precios máximos de barras para el par especificado de símbolo-período en cantidad especificada. Cabe mencionar que la cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente.



Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino, porque si la cantidad de datos resulta ser menos (o más) de la que cabe en el array, entonces se intenta redistribuirlo de tal manera para que los datos solicitados quepan íntegramente.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La propiedad del array de destino no importa, sea as_series=true o sea as_series=false. Los datos van a ser copiados de tal manera que el elemento más antiguo estará al principio de la memoria física que ha sido destinada para el array. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopyHigh(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   int              start_pos,        // posición de inicio
   int              count,            // cantidad de datos a copiar
   double           high_array[]      // array para copiar los precios máximos
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopyHigh(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   int              count,            // cantidad de datos a copiar
   double           high_array[]      // array para copiar los precios máximos
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopyHigh(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   datetime         stop_time,        // fecha y hora de finalización
   double           high_array[]      // array para copiar los precios máximos
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

high_array[]

[out] Array del tipo double.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Si el intervalo de datos solicitados se encuentra totalmente fuera del rango de datos disponibles del servidor, la función devuelve -1. Si los datos solicitados salen del rango de TERMINAL_MAXBARS (la cantidad máxima de barras en el gráfico), la función tambien devuelve -1.

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout, pero el historial seguirá cargándose y con la siguiente solicitud del mismo tipo la función devolverá más datos.

Cuando se solicitan los datos por la fecha inicial y el número de elementos requeridos, sólo se devuelven los datos cuya fecha es menor (anterior) o igual a la especificada. Este intervalo se

establece y se cuenta con la precisión de segundos. Es decir, la fecha de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre es igual o menor que la especificada.

Cuando se solicitan los datos de un período de fechas especificado, se devuelven los datos que entran sólo en este intervalo temporal especificado. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la hora de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre se encuentra en el intervalo especificado.

Por ejemplo, si el día corriente es sabado, al intentar copiar los datos del margen semanal indicando *start_time=Último_Martes* y *stop_time=Último_Viernes*, la función devolverá 0 porque la apertura en un período de tiempo semanal siempre cae en domingo, pero ninguna barra semanal no entra en el período especificado.

Si se necesita obtener el valor que corresponde a una barra corriente no finalizada, se puede usar la primera forma de llamada, indicando *start_pos*=0 y *count*=1.

Ejemplo:

```
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Un ejemplo de visualizar los valores High[i] y Low[i]"
#property description "para las barras seleccionadas de una manera aleatoria"

double High[],Low[];
//+------------------------------------------------------------------+
//| Obtenemos Low para el índice especificado de la barra            |
//+------------------------------------------------------------------+
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);
  }
//+------------------------------------------------------------------+
//| Obtenemos High para el índice especificado de la barra           |
//+------------------------------------------------------------------+
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);
  }
//+------------------------------------------------------------------+
```

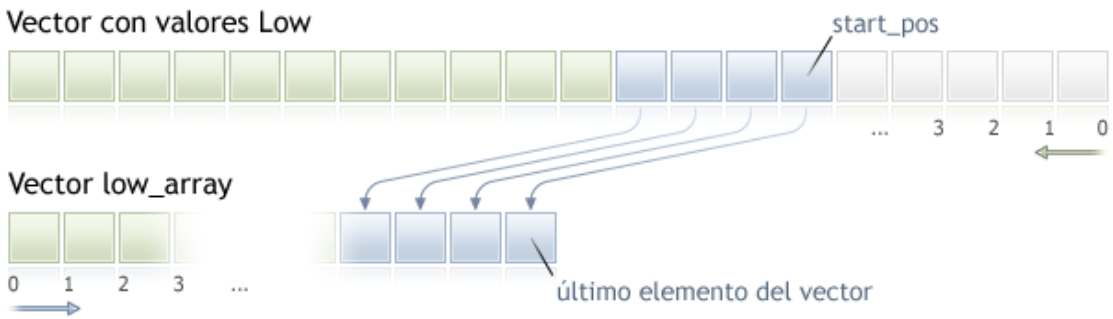
```
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- mostramos en cada tick los valores de High y Low para e la barra con índice,
//--- que equivale al segundo de llegada del tick
   datetime t=TimeCurrent();
   int sec=t%60;
   printf("High[%d] = %G  Low[%d] = %G",
          sec,iHigh(Symbol(),0,sec),
          sec,iLow(Symbol(),0,sec));
  }
```

Puede encontrar un ejemplo más detallado de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes estilos de construcción:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyLow

La función recibe en el array low_array datos históricos de los precios mínimos de barras para el par especificado de símbolo-período en cantidad especificada. Cabe mencionar que la cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente.



Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino, porque si la cantidad de datos resulta ser menos (o más) de la que cabe en el array, entonces se intenta redistribuirlo de tal manera para que los datos solicitados quepan íntegramente.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La propiedad del array de destino no importa, sea as_series=true o sea as_series=false. Los datos van a ser copiados de tal manera que el elemento más antiguo estará al principio de la memoria física que ha sido destinada para el array. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopyLow(
   string           symbol_name,     // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,       // período
   int              start_pos,       // posición de inicio
   int              count,           // cantidad de datos a copiar
   double           low_array[]      // array para copiar los precios mínimos
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopyLow(
   string           symbol_name,     // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,       // período
   datetime         start_time,      // fecha y hora de inicio
   int              count,           // cantidad de datos a copiar
   double           low_array[]      // array para copiar los precios mínimos
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopyLow(
   string           symbol_name,     // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,       // período
   datetime         start_time,      // fecha y hora de inicio
   datetime         stop_time,       // fecha y hora de finalización
   double           low_array[]      // array para copiar los precios mínimos
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

low_array[]

[out] Array del tipo double.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Si el intervalo de datos solicitados se encuentra totalmente fuera del rango de datos disponibles del servidor, la función devuelve -1. Si los datos solicitados salen del rango de TERMINAL_MAXBARS (la cantidad máxima de barras en el gráfico), la función tambien devuelve -1.

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout, pero el historial seguirá cargándose y con la siguiente solicitud del mismo tipo la función devolverá más datos.

Cuando se solicitan los datos por la fecha inicial y el número de elementos requeridos, sólo se devuelven los datos cuya fecha es menor (anterior) o igual a la especificada. Este intervalo se

establece y se cuenta con la precisión de segundos. Es decir, la fecha de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre es igual o menor que la especificada.

Cuando se solicitan los datos de un período de fechas especificado, se devuelven los datos que entran sólo en este intervalo temporal especificado. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la hora de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre se encuentra en el intervalo especificado.

Por ejemplo, si el día corriente es sabado, al intentar copiar los datos del margen semanal indicando *start_time=Último_Martes* y *stop_time=Último_Viernes*, la función devolverá 0 porque la apertura en un período de tiempo semanal siempre cae en domingo, pero ninguna barra semanal no entra en el período especificado.

Si se necesita obtener el valor que corresponde a una barra corriente no finalizada, se puede usar la primera forma de llamada, indicando *start_pos*=0 y *count*=1.

Puede encontrar un ejemplo de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes estilos de construcción:

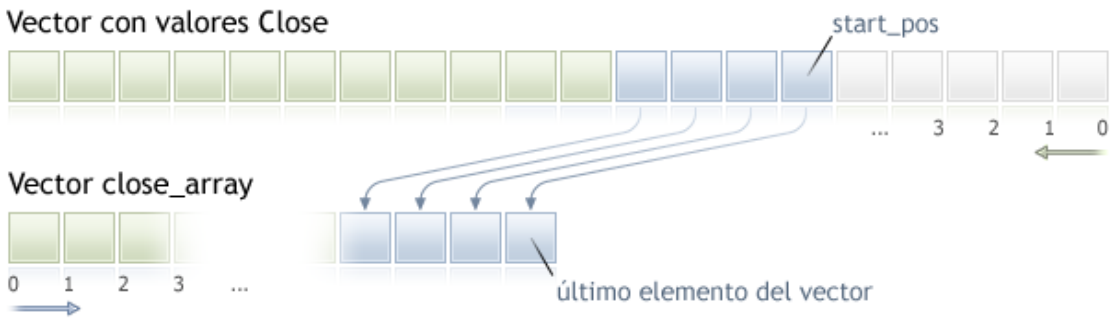
- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

Véase también

CopyHigh

CopyClose

La función recibe en el array close_array datos históricos de los precios del cierre de barras para el par especificado de símbolo-período en cantidad especificada. Cabe mencionar que la cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente.



Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino, porque si la cantidad de datos resulta ser menos (o más) de la que cabe en el array, entonces se intenta redistribuirlo de tal manera para que los datos solicitados quepan íntegramente.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La propiedad del array de destino no importa, sea as_series=true o sea as_series=false. Los datos van a ser copiados de tal manera que el elemento más antiguo estará al principio de la memoria física que ha sido destinada para el array. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopyClose(
   string           symbol_name,       // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,         // período
   int              start_pos,         // posición de inicio
   int              count,             // cantidad de datos a copiar
   double           close_array[]      // array para copiar los precios de cierre
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopyClose(
   string           symbol_name,       // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,         // período
   datetime         start_time,        // fecha y hora de inicio
   int              count,             // cantidad de datos a copiar
   double           close_array[]      // array para copiar los precios de cierre
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopyClose(
   string           symbol_name,       // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,         // período
   datetime         start_time,        // fecha y hora de inicio
   datetime         stop_time,         // fecha y hora de finalización
   double           close_array[]      // array para copiar los precios de cierre
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

close_array[]

[out] Array del tipo double.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Si el intervalo de datos solicitados se encuentra totalmente fuera del rango de datos disponibles del servidor, la función devuelve -1. Si los datos solicitados salen del rango de TERMINAL_MAXBARS (la cantidad máxima de barras en el gráfico), la función tambien devuelve -1.

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout, pero el historial seguirá cargándose y con la siguiente solicitud del mismo tipo la función devolverá más datos.

Cuando se solicitan los datos por la fecha inicial y el número de elementos requeridos, sólo se devuelven los datos cuya fecha es menor (anterior) o igual a la especificada. Este intervalo se

establece y se cuenta con la precisión de segundos. Es decir, la fecha de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre es igual o menor que la especificada.

Cuando se solicitan los datos de un período de fechas especificado, se devuelven los datos que entran sólo en este intervalo temporal especificado. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la hora de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre se encuentra en el intervalo especificado.

Por ejemplo, si el día corriente es sabado, al intentar copiar los datos del margen semanal indicando *start_time=Último_Martes* y *stop_time=Último_Viernes*, la función devolverá 0 porque la apertura en un período de tiempo semanal siempre cae en domingo, pero ninguna barra semanal no entra en el período especificado.

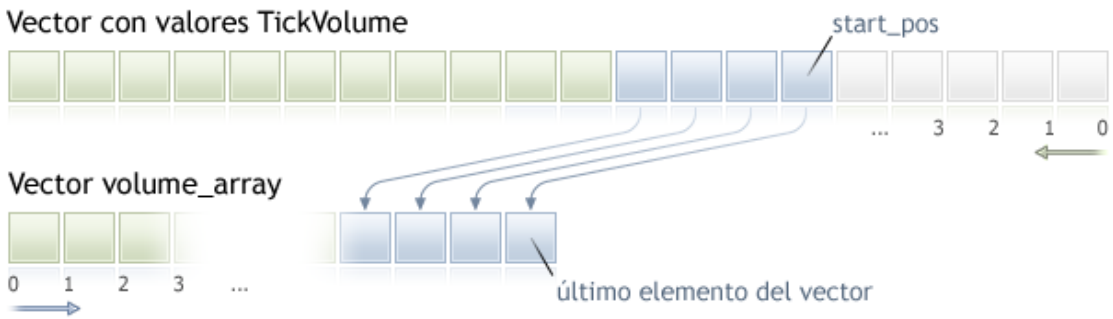
Si se necesita obtener el valor que corresponde a una barra corriente no finalizada, se puede usar la primera forma de llamada, indicando *start_pos*=0 y *count*=1.

Puede encontrar un ejemplo de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes estilos de construcción:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyTickVolume

La función recibe en el array volume_array datos históricos de los volúmenes de tick para el par especificado de símbolo-período en cantidad especificada. Cabe mencionar que la cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente.



Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino, porque si la cantidad de datos resulta ser menos (o más) de la que cabe en el array, entonces se intenta redistribuirlo de tal manera para que los datos solicitados quepan íntegramente.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La propiedad del array de destino no importa, sea as_series=true o sea as_series=false. Los datos van a ser copiados de tal manera que el elemento más antiguo estará al principio de la memoria física que ha sido destinada para el array. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopyTickVolume(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   int              start_pos,        // posición de inicio
   int              count,            // cantidad de datos a copiar
   long             volume_array[]    // array para copiar los volúmenes de tick
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopyTickVolume(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   int              count,            // cantidad de datos a copiar
   long             volume_array[]    // array para copiar los volúmenes de tick
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopyTickVolume(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   datetime         stop_time,        // fecha y hora de finalización
   long             volume_array[]    // array para copiar los volúmenes de tick
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

volume_array[]

[out] Array del tipo long.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Si el intervalo de datos solicitados se encuentra totalmente fuera del rango de datos disponibles del servidor, la función devuelve -1. Si los datos solicitados salen del rango de TERMINAL_MAXBARS (la cantidad máxima de barras en el gráfico), la función tambien devuelve -1.

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout, pero el historial seguirá cargándose y con la siguiente solicitud del mismo tipo la función devolverá más datos.

Cuando se solicitan los datos por la fecha inicial y el número de elementos requeridos, sólo se devuelven los datos cuya fecha es menor (anterior) o igual a la especificada. Este intervalo se

establece y se cuenta con la precisión de segundos. Es decir, la fecha de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre es igual o menor que la especificada.

Cuando se solicitan los datos de un período de fechas especificado, se devuelven los datos que entran sólo en este intervalo temporal especificado. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la hora de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre se encuentra en el intervalo especificado.

Por ejemplo, si el día corriente es sabado, al intentar copiar los datos del margen semanal indicando *start_time=Último_Martes* y *stop_time=Último_Viernes*, la función devolverá 0 porque la apertura en un período de tiempo semanal siempre cae en domingo, pero ninguna barra semanal no entra en el período especificado.

Si se necesita obtener el valor que corresponde a una barra corriente no finalizada, se puede usar la primera forma de llamada, indicando *start_pos*=0 y *count*=1.

Ejemplo:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   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[],
```

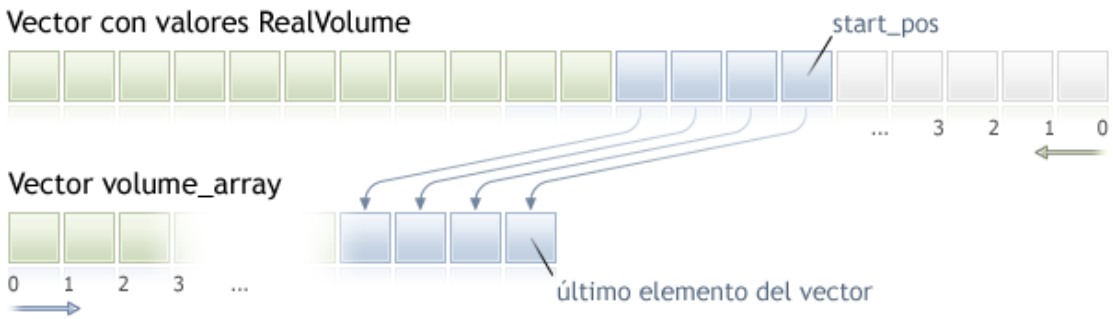
```
                const double &low[],
                const double &close[],
                const long &tick_volume[],
                const long &volume[],
                const int &spread[])
  {
//---
   if(prev_calculated==0)
     {
      long timeseries[];
      ArraySetAsSeries(timeseries,true);
      int prices=CopyTickVolume(Symbol(),0,0,bars,timeseries);
      for(int i=0;i<rates_total-prices;i++) TickVolumeBuffer[i]=0.0;
      for(int i=0;i<prices;i++) TickVolumeBuffer[rates_total-1-i]=timeseries[prices-1
      Print("Recibida la siguiente cantidad de valores históricos TickVolume: "+price
     }
   else
     {
      long timeseries[];
      int prices=CopyTickVolume(Symbol(),0,0,1,timeseries);
      TickVolumeBuffer[rates_total-1]=timeseries[0];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Puede encontrar un ejemplo más detallado de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes estilos de construcción:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopyRealVolume

La función recibe en el array volume_array datos históricos de los volúmenes comerciales para el par especificado de símbolo-período en cantidad especificada. Cabe mencionar que la cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente.



Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino, porque si la cantidad de datos resulta ser menos (o más) de la que cabe en el array, entonces se intenta redistribuirlo de tal manera para que los datos solicitados quepan íntegramente.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La propiedad del array de destino no importa, sea as_series=true o sea as_series=false. Los datos van a ser copiados de tal manera que el elemento más antiguo estará al principio de la memoria física que ha sido destinada para el array. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopyRealVolume(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   int              start_pos,        // posición de inicio
   int              count,            // cantidad de datos a copiar
   long             volume_array[]    // array para copiar los volúmenes
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopyRealVolume(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   int              count,            // cantidad de datos a copiar
   long             volume_array[]    // array para copiar los volúmenes
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopyRealVolume(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   datetime         stop_time,        // fecha y hora de finalización
   long             volume_array[]    // array para copiar los volúmenes
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

volume_array[]

[out] Array del tipo long.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Si el intervalo de datos solicitados se encuentra totalmente fuera del rango de datos disponibles del servidor, la función devuelve -1. Si los datos solicitados salen del rango de TERMINAL_MAXBARS (la cantidad máxima de barras en el gráfico), la función tambien devuelve -1.

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout, pero el historial seguirá cargándose y con la siguiente solicitud del mismo tipo la función devolverá más datos.

Cuando se solicitan los datos por la fecha inicial y el número de elementos requeridos, sólo se devuelven los datos cuya fecha es menor (anterior) o igual a la especificada. Este intervalo se

establece y se cuenta con la precisión de segundos. Es decir, la fecha de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre es igual o menor que la especificada.

Cuando se solicitan los datos de un período de fechas especificado, se devuelven los datos que entran sólo en este intervalo temporal especificado. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la hora de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre se encuentra en el intervalo especificado.

Por ejemplo, si el día corriente es sabado, al intentar copiar los datos del margen semanal indicando *start_time=Último_Martes* y *stop_time=Último_Viernes*, la función devolverá 0 porque la apertura en un período de tiempo semanal siempre cae en domingo, pero ninguna barra semanal no entra en el período especificado.

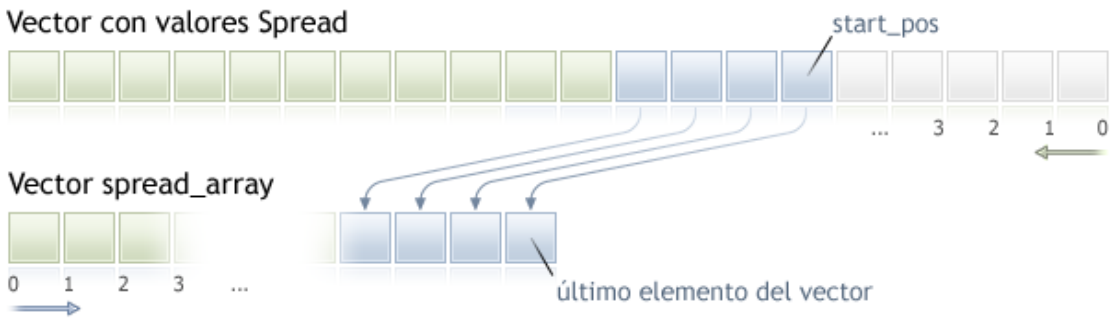
Si se necesita obtener el valor que corresponde a una barra corriente no finalizada, se puede usar la primera forma de llamada, indicando *start_pos*=0 y *count*=1.

Puede encontrar un ejemplo de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes estilos de construcción:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

CopySpread

La función recibe en el array spread_array datos históricos de los spreads para el par especificado de símbolo-período en cantidad especificada. Cabe mencionar que la cuenta de elementos desde la posición de inicio se realiza del presente al pasado, es decir, la posición de inicio igual a 0 significa la barra corriente.



Cuando copiamos los datos sin conocer todavía el volumen a copiar, se recomienda usar un array dinámico como array de destino, porque si la cantidad de datos resulta ser menos (o más) de la que cabe en el array, entonces se intenta redistribuirlo de tal manera para que los datos solicitados quepan íntegramente.

Si sabemos la cantidad de datos que tenemos que copiar, con el fin de evitar la asignación excesiva de memoria es mejor hacerlo a un buffer asignado estáticamente.

La propiedad del array de destino no importa, sea as_series=true o sea as_series=false. Los datos van a ser copiados de tal manera que el elemento más antiguo estará al principio de la memoria física que ha sido destinada para el array. Existen 3 variantes de la función.

Llamada según la posición de inicio y el número de elementos requeridos

```
int  CopySpread(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   int              start_pos,        // posición de inicio
   int              count,            // cantidad de datos a copiar
   int              spread_array[]    // array para copiar los spreads
   );
```

Llamada según la fecha de inicio y el número de elementos requeridos

```
int  CopySpread(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   int              count,            // cantidad de datos a copiar
   int              spread_array[]    // array para copiar los spreads
   );
```

Llamada según la fecha de inicio y finalización del intervalo de tiempo requerido

```
int  CopySpread(
   string           symbol_name,      // nombre del símbolo
   ENUM_TIMEFRAMES  timeframe,        // período
   datetime         start_time,       // fecha y hora de inicio
   datetime         stop_time,        // fecha y hora de finalización
   int              spread_array[]    // array para copiar los spreads
   );
```

Parámetros

symbol_name

[in] Símbolo.

timeframe

[in] Período.

start_pos

[in] Posición del primer elemento a copiar.

count

[in] Cantidad de elementos a copiar.

start_time

[in] Hora de la barra correspondiente al primer elemento.

stop_time

[in] Hora de la barra correspondiente al último elemento.

spread_array[]

[out] Array del tipo int.

Valor devuelto

Cantidad de elementos copiados del array, o -1 en caso del error.

Nota

Si el intervalo de datos solicitados se encuentra totalmente fuera del rango de datos disponibles del servidor, la función devuelve -1. Si los datos solicitados salen del rango de TERMINAL_MAXBARS (la cantidad máxima de barras en el gráfico), la función tambien devuelve -1.

Al solicitar los datos del indicador, si las series temporales solicitadas todavía no están construidas o hay que bajarlas del servidor, la función devolverá inmediatamente -1. Aunque en este caso, se iniciará el proceso de descarga/construcción.

Cuando se solicitan los datos a un Asesor Experto o un script, se iniciará la descarga desde el servidor si el terminal no dispone de estos datos a nivel local, o se empezará la construcción de la serie temporal necesaria si se puede construir los datos usando el historial local y ellos todavía no están listos. La función devolverá aquella cantidad de datos que estarán listos para el momento de vencimiento de timeout, pero el historial seguirá cargándose y con la siguiente solicitud del mismo tipo la función devolverá más datos.

Cuando se solicitan los datos por la fecha inicial y el número de elementos requeridos, sólo se devuelven los datos cuya fecha es menor (anterior) o igual a la especificada. Este intervalo se

establece y se cuenta con la precisión de segundos. Es decir, la fecha de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre es igual o menor que la especificada.

Cuando se solicitan los datos de un período de fechas especificado, se devuelven los datos que entran sólo en este intervalo temporal especificado. Este intervalo se establece y se cuenta con la precisión de segundos. Es decir, la hora de apertura de cualquier barra para la que se devuelve el valor (volumen, spread, valor en el búfer de indicador, precio Open, High, Low, Close o la hora de apertura Time) siempre se encuentra en el intervalo especificado.

Por ejemplo, si el día corriente es sabado, al intentar copiar los datos del margen semanal indicando *start_time=Último_Martes* y *stop_time=Último_Viernes*, la función devolverá 0 porque la apertura en un período de tiempo semanal siempre cae en domingo, pero ninguna barra semanal no entra en el período especificado.

Si se necesita obtener el valor que corresponde a una barra corriente no finalizada, se puede usar la primera forma de llamada, indicando *start_pos*=0 y *count*=1.

Ejemplo:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Spread
#property indicator_label1  "Spread"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      bars=3000;
//--- indicator buffers
double         SpreadBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,SpreadBuffer,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)
     {
      int spread_int[];
      ArraySetAsSeries(spread_int,true);
      int spreads=CopySpread(Symbol(),0,0,bars,spread_int);
      Print("Recibida la siguiente cantidad de valores históricos del spread: ",sprea
      for (int i=0;i<spreads;i++)
      {
      SpreadBuffer[rates_total-1-i]=spread_int[i];
      if(i<=30) Print("spread["+i+"] = ",spread_int[i]);
      }
     }
   else
     {
      double Ask,Bid;
      Ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
      Bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
      Comment("Ask = ",Ask,"  Bid = ",Bid);
      SpreadBuffer[rates_total-1]=(Ask-Bid)/Point();
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Puede encontrar un ejemplo más detallado de la solicitud de datos históricos en la sección Modos de enlace de objetos. En el script de aquella sección se muestra cómo se puede obtener los valores del indicador iFractals sobre las últimas 1000 barras, y cómo luego se puede visualizar en el gráfico los diez últimos fractales arriba y los diez último fractales abajo. Esta técnica puede ser utilizada para todos los indicadores que tienen omisiones de valores y suelen realizarse utilizando los siguientes estilos de construcción:

- DRAW_SECTION,
- DRAW_ARROW,
- DRAW_ZIGZAG,
- DRAW_COLOR_SECTION,
- DRAW_COLOR_ARROW,
- DRAW_COLOR_ZIGZAG.

Operaciones con gráficos

Estas son las funciones para trabajar con los gráficos. Todas las operaciones con gráficos están disponibles únicamente en los Asesores Expertos y en los scripts.

Función	Acción
ChartApplyTemplate	Aplica al gráfico especificado una plantilla desde un archivo especificado
ChartSaveTemplate	Guarda los ajustes actuales del gráfico en una plantilla con el nombre especificado
ChartWindowFind	Devuelve el número de una subventana en la que se encuentra el indicador
ChartTimePriceToXY	Convierte las coordenadas del gráfico desde la representación hora/precio a las coordinadas en el eje X y Y
ChartXYToTimePrice	Convierte las coordenadas X y Y del gráfico a los valores hora y precio
ChartOpen	Abre un gráfico nuevo con un símbolo y período especificados
ChartClose	Cierra un gráfico especificado
ChartFirst	Devuelve el identificador del primer gráfico del terminal de cliente
ChartNext	Devuelve el identificador del gráfico que sigue después del especificado
ChartSymbol	Devuelve el nombre del símbolo del gráfico especificado
ChartPeriod	Devuelve el valor del período del gráfico especificado
ChartRedraw	Activa el redibujo forzado de un gráfico especificado
ChartSetDouble	Establece el valor del tipo double de una propiedad correspondiente del gráfico especificado
ChartSetInteger	Establece el valor del tipo entero (datetime, int, color, bool o char) de una propiedad correspondiente del gráfico especificado
ChartSetString	Establece el valor del tipo string de una propiedad correspondiente del gráfico especificado
ChartGetDouble	Devuelve el valor de una propiedad correspondiente del gráfico especificado

ChartGetInteger	Devuelve el valor del tipo entero de una propiedad correspondiente del gráfico especificado
ChartGetString	Devuelve el valor literal de una propiedad correspondiente del gráfico especificado
ChartNavigate	Desplaza el gráfico especificado a una cantidad de barras especificada respecto a la posición del gráfico indicada
ChartID	Devuelve el identificador del gráfico corriente
ChartIndicatorAdd	Añade un indicador con el manejador (handle) especificado a la ventana del gráfico especificado
ChartIndicatorDelete	Quita el indicador con el nombre especificado de la ventana del gráfico especificada
ChartIndicatorGet	Devuelve el manejador del indicador con el nombre corto especificado en la ventana del gráfico especificada
ChartIndicatorName	Devuelve el nombre breve del indicador según su número en la lista de indicadores en la determinada ventana del gráfico
ChartIndicatorsTotal	Devuelve el número de todos los indicadores vinculados con la determinada ventana del gráfico
ChartWindowOnDropped	Devuelve el número de la subventana del gráfico a la que el Asesor Experto, script, objeto o indicador ha sido arrastrado con el ratón
ChartPriceOnDropped	Devuelve la coordinada de precios que corresponde al punto al que el Asesor Experto o script ha sido arrastrado con el ratón
ChartTimeOnDropped	Devuelve la coordinada de tiempo que corresponde al punto al que el Asesor Experto o script ha sido arrastrado con el ratón
ChartXOnDropped	Devuelve la coordinada del eje de X que corresponde al punto al que el Asesor Experto o script ha sido arrastrado con el ratón
ChartYOnDropped	Devuelve la coordinada del eje de Y que corresponde al punto al que el Asesor Experto o script ha sido arrastrado con el ratón
ChartSetSymbolPeriod	Cambia el valor del símbolo y período del gráfico especificado
ChartScreenShot	Hace un screenshot del gráfico especificado en

	formato gif

ChartApplyTemplate

Aplica al gráfico una plantilla especificada.

```
bool  ChartApplyTemplate(
   long          chart_id,      // identificador del gráfico
   const string  filename       // nombre del archivo con la plantilla
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

filename

[in] Nombre del archivo que contiene la plantilla.

Valor devuelto

Si la plantilla se aplica con éxito, la función devuelve true, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Si a través de esta función desde el Asesor Experto se cargará una plantilla nueva en el gráfico al que este Asesor Experto está adjuntado, entonces este Asesor Experto será descargado y no podrá continuar su trabajo.

Uso de plantillas

Los recursos del lenguaje MQL5 permiten establecer varias propiedades del gráfico, comprendido entre ellas los colores, utilizando la función ChartSetInteger() :

- Color del fondo del gráfico;
- Color de ejes, escala y línea OHLC;
- Color de cuadrícula;
- Color de volúmenes y niveles de apertura de posiciones;
- Color de la barra arriba, sombra y borde del cuerpo de vela alcista;
- Color de la barra abajo, sombra y borde del cuerpo de vela bajista;
- Color de la línea del gráfico y velas japonesas "Doji";
- Color del cuerpo de vela alcista;
- Color del cuerpo de vela bajista;
- Color de la línea del precio Bid;
- Color de la línea del precio Ask;
- Color de la línea del precio de la última transacción realizada (Last);
- Color de niveles de órdenes Stop (Stop Loss y Take Profit).

Además, en el gráfico pueden haber varios objetos gráficos e indicadores. Será suficiente configurar una vez la apariencia del gráfico, dotándolo con todos los indicadores necesarios, y guardarlo como

una plantilla para luego poder aplicar esta plantilla a cualquier gráfico.

La función ChartApplyTemplate() sirve para usar las plantillas guardadas anteriormente. Esta función se puede utilizar en cualquier programa mql5. La ruta del archivo en el que se encuentra la plantilla se pasa con el segundo parámetro de la función ChartApplyTemplate(). La ruta del archivo en el que se encuentra la plantilla se pasa. La búsqueda del archivo de una plantilla se realiza según las siguientes reglas:

- si la ruta se empieza con la barra inversa separadora "\" (se escribe "\\"), entonces la plantilla se busca según la ruta catálogo_de_datos_del_terminal\MQL5,
- si no hay ninguna barra inversa, la plantilla se busca respecto a la ubicación del archivo ejecutable EX5 en el que se realiza la llamada a la función ChartApplyTemplate();
- si la plantilla no ha sido encontrada en las primeras dos opciones, la búsqueda continúa en la carpeta directorio_del_terminal\Profiles\Templates\.

Aquí directorio_del_terminal significa la carpeta desde la que ha sido iniciado el terminal de cliente MetaTrader 5, mientras que catálogo_de_datos_del_terminal significa la carpeta en la que se guardan los archivos editables y su ubicación puede depender del tipo del sistema operativo, nombre del usuario y las configuraciones de seguridad del ordenador. Por lo general, se trata de diferentes carpetas, aunque en algunas ocasiones pueden coincidir.

La ubicación de las carpetas catálogo_de_datos_del_terminal y directorio_del_terminal se puede averiguar a través de la función TerminalInfoString().

```
//--- carpeta desde la que ha sido iniciado el terminal
   string terminal_path=TerminalInfoString(TERMINAL_PATH);
   Print("Directorio del terminal:",terminal_path);
//--- carpeta de datos del terminal que contiene la carpeta MQL5 con los EAs e indica
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
   Print("Carpeta de datos del terminal:",terminal_data_path);
```

Ejemplos de entrada:

```
//--- buscamos la plantilla en la carpeta catálogo_de_datos_del_terminal\MQL5\
ChartApplyTemplate(0,"\\first_template.tpl"))

//--- buscamos la plantilla en la carpeta catálogo_del_archivo_ejecutable_EX5\, luego
ChartApplyTemplate(0,"second_template.tpl"))

//--- buscamos la plantilla en la carpeta catálogo_del_archivo_ejecutable_EX5\My_temp
ChartApplyTemplate(0,"My_templates\\third_template.tpl"))
```

Las plantillas no pertenecen a los recursos, y no se puede insertarlos en un archivo ejecutable EX5.

Ejemplo:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- vamos a intentar abrir y aplicar la plantilla my_template.tpl
   if(!ChartApplyTemplate(0,"my_template"))
     {
      //--- no se ha podido aplicar la plantilla my_template
      Print("Fallo al aplicar la plantilla 'my_template', error ",GetLastError());
      ResetLastError();
      //--- vamos a buscar simplemente el archivo "my_template" sin extensión tpl
      if(FileIsExist("my_template"))
         Print("Existe el archivo my_template, pero se necesita 'my_template.tpl'");
      else
         Print("El archivo 'my_template.tpl' no ha sido encontrado en la carpeta "
               +TerminalInfoString(TERMINAL_PATH)
               +"\\MQL5\\Files");

      //--- Ahora pasamos correctamente el nombre del archivo con la extensión *.tpl
      if(ChartApplyTemplate(0,"my_template.tpl"))
         Print("La plantilla 'my_template.tpl' se ha aplicado con éxito");
      else
         Print("Fallo al aplicar la plantilla 'my_template.tpl', error ",GetLastError

     }
   else
     {
      //--- la plantilla se guarda en el archivo sin extensión
      Print("La plantilla 'my_template' se ha aplicado con éxito - archivo sin extens
     }
  }
```

Véase también

Recursos

ChartSaveTemplate

Guarda los ajustes actuales del gráfico en una plantilla con el nombre especificado.

```
bool  ChartSaveTemplate(
   long          chart_id,     // identificador del gráfico
   const string  filename      // nombre del archivo para guardar la plantilla
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

filename

[in] Nombre del archivo para guardar la plantilla. La extensión ".tpl" se añade al nombre del archivo de forma automática, no hace falta ponerla. La plantilla se guarda en la carpeta **directorio_del_terminal**\Profiles\Templates\, y puede ser utilizada también para la aplicación manual en el terminal. Si ya existe una plantilla con el mismo nombre, entonces su contenido será sobrescrito de nuevo.

Valor devuelto

En caso del éxito la función devuelve true, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Una plantilla permite guardar los ajustes del gráfico con todos los indicadores y objetos gráficos aplicados a él para que luego el usuario pueda aplicarla al otro gráfico.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                      Test_ChartSaveTemplate.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property script_show_inputs
//--- input parameters
input string               symbol="GBPUSD";  // símbolo del gráfico nuevo
input ENUM_TIMEFRAMES      period=PERIOD_H3; // período de tiempo del gráfico nuevo
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- primero adjuntamos al gráfico los indicadores
   int handle;
```

```
//--- preparamos los indicadores para el uso
   if(!PrepareZigzag(NULL,0,handle)) return; // fallo, entonces salimos
//--- colocamos el indicador en el gráfico actual, pero... en otra ventana :)
   if(!ChartIndicatorAdd(0,1,handle))
     {
      PrintFormat("Fallo al adjuntar al gráfico %s/%s el indicador con el manejador=%
                  _Symbol,
                  EnumToString(_Period),
                  handle,
                  GetLastError());
      //--- finalizamos anticipadamente el trabajo del programa
      return;
     }
//--- actualizamos el gráfico para ver el indicador colocado
   ChartRedraw();
//--- encontraremos los dos últimos virajes del zigzag
   double two_values[];
   datetime two_times[];
   if(!GetLastTwoFractures(two_values,two_times,handle))
     {
      PrintFormat("No se ha podido encontrar los dos últimos virajes en el indicador
      //--- finalizamos anticipadamente el trabajo del programa
      return;
     }
//--- ahora adjuntamos el canal de desviación estándar
   string channel="StdDeviation Channel";
   if(!ObjectCreate(0,channel,OBJ_STDDEVCHANNEL,0,two_times[1],0))
     {
      PrintFormat("Fallo al crear el objeto %s. Código del error %d",
                  EnumToString(OBJ_STDDEVCHANNEL),GetLastError());
      return;
     }
   else
     {
      //--- el canal está creado, definimos el segundo punto de apoyo
      ObjectSetInteger(0,channel,OBJPROP_TIME,1,two_times[0]);
      //--- asignaremos al canal el texto de ayuda emergente
      ObjectSetString(0,channel,OBJPROP_TOOLTIP,"Demo from MQL5 Help");
      //--- actualizaremos el gráfico
      ChartRedraw();
     }
//--- guardaremos toda esta maravilla en la plantilla
   ChartSaveTemplate(0,"StdDevChannelOnZigzag");
//--- abriremos nuevo gráfico y aplicaremos la plantilla guardada
   long new_chart=ChartOpen(symbol,period);
   //--- activaremos la opción de descripción emergente para los objetos gráficos
   ChartSetInteger(new_chart,CHART_SHOW_OBJECT_DESCR,true);
   if(new_chart!=0)
```

```
      {
       //--- aplicaremos al gráfico nuevo la plantilla guardada
       ChartApplyTemplate(new_chart,"StdDevChannelOnZigzag");
      }
    Sleep(10000);
   }
//+------------------------------------------------------------------+
//| crea el manejador del zigzag y asegura la disponibilidad de sus datos       |
//+------------------------------------------------------------------+
bool PrepareZigzag(string sym,ENUM_TIMEFRAMES tf,int &h)
   {
    ResetLastError();
//--- el indicador Zigzag debe ubicarse en la carpeta catálogo_de_datos_del_terminal\I
    h=iCustom(sym,tf,"Examples\\Zigzag");
    if(h==INVALID_HANDLE)
      {
       PrintFormat("%s: Fallo al crear el manejador del indicador Zigzag. Código del e
                   __FUNCTION__,GetLastError());
       return false;
      }
//--- cuando se crea el manejador del indicador, él necesita tiempo para calcular los
    int k=0; // número de intentos para conseguir la calculación del indicador
//--- esperamos el cálculo en el ciclo, hacemos una pausa de 50 millisegundos si el c
    while(BarsCalculated(h)<=0)
      {
       k++;
       //--- mostramos el número de intentos
       PrintFormat("%s: k=%d",__FUNCTION__,k);
       //--- esperamos 50 millisegundos hasta que el indicador se calcule
       Sleep(50);
       //--- si han sido más de 100 intentos, algo va mal
       if(k>100)
         {
          //--- avisamos sobre el problema
          PrintFormat("No se ha podido calcular el indicador con %d intentos!");
          //--- finalizamos anticipadamente el trabajo del programa
          return false;
         }
      }
//--- todo está listo, el indicador está creado y los valores están calculados
    return true;
   }
//+------------------------------------------------------------------+
//|  busca los dos últimos virajes del zigzag y coloca en los arrays        |
//+------------------------------------------------------------------+
bool GetLastTwoFractures(double &get_values[],datetime &get_times[],int handle)
   {
    double values[];          // array para obtener los valores del zigzag
```

```
   datetime times[];         // array para obtener el tiempo
   int size=100;             // tamaño de arrays
   ResetLastError();
//--- copiamos los 100 últimos valores del indicador
   int copied=CopyBuffer(handle,0,0,size,values);
//--- comprobamos cuántos se han copiado
   if(copied<100)
     {
      PrintFormat("%s: No se han copiado %d valores del indicador con el manejador=%d
                  __FUNCTION__,size,handle,GetLastError());
      return false;
     }
//--- definimos el orden de acceso al array como en una serie temporal
   ArraySetAsSeries(values,true);
//--- escribiremos aquí los números de las barras en las que se han encontrado los vi
   int positions[];
//--- fijaremos los tamaños de arrays
   ArrayResize(get_values,3); ArrayResize(get_times,3); ArrayResize(positions,3);
//--- contadores
   int i=0,k=0;
//--- empezamos a buscar los virajes
   while(i<100)
     {
      double v=values[i];
      //--- los valores vacíos no nos interesan
      if(v!=0.0)
        {
         //--- recordamos el número de la barra
         positions[k]=i;
         //--- recordamos el valor del zigzag en el viraje
         get_values[k]=values[i];
         PrintFormat("%s: Zigzag[%d]=%G",__FUNCTION__,i,values[i]);
         //--- aumentamos contadores
         k++;
         //--- si hemos encontrado dos virajes, rompemos el ciclo
         if(k>2) break;
        }
      i++;
     }
//--- definimos el orden de acceso a los arrays como en una serie temporal
   ArraySetAsSeries(times,true);   ArraySetAsSeries(get_times,true);
   if(CopyTime(_Symbol,_Period,0,size,times)<=0)
     {
      PrintFormat("%s: Fallo al copiar %d valores desde CopyTime(). Código del error
                  __FUNCTION__,size,GetLastError());
      return false;
     }
//--- encontraremos el tiempo de apertura de las barras en las que han sido encontrad
```

```
   get_times[0]=times[positions[1]];// el penúltimo valor se escribirá como el primer
   get_times[1]=times[positions[2]];// el tercer valor desde el final será el segundo
   PrintFormat("%s: el primero=%s,  el segundo=%s",__FUNCTION__,TimeToString(get_time
//--- todo ha salido bien
   return true;
  }
```

Véase también

ChartApplyTemplate(), Recursos

ChartWindowFind

Devuelve el número de una subventana en la que se encuentra el indicador. Existen 2 variantes de la función.

1. La función busca en el gráfico especificado la subventana con el "nombre breve" del indicador (el nombre breve se muestra en la parte superior izquierda de la subventana), y en caso del éxito devuelve el número de subventana.

```
int  ChartWindowFind(
   long     chart_id,                  // identificador del gráfico
   string   indicator_shortname        // nombre breve del indicador, véase INDICATOR
```

2. La función debe invocarse desde el indicador personalizado y devuelve el número de la subventana en la que este indicador trabaja.

```
int  ChartWindowFind();
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

indicator_shortname

[in] Nombre breve del indicador.

Valor devuelto

Devuelve el número de subventana en caso del éxito. Cero significa la ventana principal del gráfico. En caso de fallo devuelve -1.

Nota

Si se llama a la segunda variante de la función (sin parámetros) desde un script o Asesor Experto, devuelve -1.

No se debe confundir el nombre breve del indicador con el nombre del archivo que se indica durante la creación del indicador por las funciones iCustom() y IndicatorCreate(). Si el nombre breve del indicador no se especifica de manera explícita, entonces durante la compilación ahí se indica el nombre del archivo con el código fuente del indicador.

Hay que formar correctamente el nombre breve del indicador que se escribe en la propiedad INDICATOR_SHORTNAME mediante la función IndicatorSetString(). Es recomendable que el nombre breve contenga los valores de los parámetros de entrada del indicador, puesto que en la función ChartIndicatorDelete() la identificación del indicador que se quita del gráfico se realiza precisamente por el nombre breve.

Ejemplo:

```
#property script_show_inputs
//--- input parameters
input string   shortName="MACD(12,26,9)";
//+------------------------------------------------------------------+
//|  devuelve el número de la ventana del gráfico con este indicador |
```

```
//+------------------------------------------------------------------+
int GetIndicatorSubWindowNumber(long chartID=0,string short_name="")
  {
   int window=-1;
//---
   if((ENUM_PROGRAM_TYPE)MQL5InfoInteger(MQL5_PROGRAM_TYPE)==PROGRAM_INDICATOR)
     {
      //--- la función ha sido invocada desde el indicador, no hace falta el nombre
      window=ChartWindowFind();
     }
   else
     {
      //--- la función ha sido invocada desde el Asesor Experto o 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("Indicador "+shortName+" se encuentra en la ventana #"+(string)window);
   else
      Print("Indicador "+shortName+" no ha sido encontrado. window = "+(string)window
  }
```

Véase también

ObjectCreate(), ObjectFind()

ChartTimePriceToXY

Convierte las coordenadas del gráfico desde la representación hora/precio a las coordinadas en el eje X y Y.

```
bool  ChartTimePriceToXY(
   long           chart_id,      // identificador del gráfico
   int            sub_window,    // número de subventana
   datetime       time,          // fecha/hora en el gráfico
   double         price,         // precio en el gráfico
   int&           x,             // coordinada X para la hora en el gráfico
   int&           y              // coordinada Y para el precio en el gráfico
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

sub_window

[in] Número de subventana del gráfico. 0 significa la ventana principal del gráfico.

time

[in] Valor de la hora en el gráfico para el cual se recibirá el valor en píxeles en el eje X. El inicio de las coordinadas se encuentra en la esquina superior izquierda de la ventana/subventana.

price

[in] Valor del precio en el gráfico para el cual se recibirá el valor en píxeles en el eje Y. El inicio de las coordinadas se encuentra en la esquina superior izquierda de la ventana/subventana.

x

[out] Variable en la que se recibirá la conversión de la hora a la coordinada X.

y

[out] Variable en la que se recibirá la conversión del precio a la coordinada Y.

Valor devuelto

Devuelve true en caso de la ejecución con éxito, de lo contrario - false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Véase también

ChartXYToTimePrice()

ChartXYToTimePrice

Convierte las coordenadas X y Y del gráfico a los valores hora y precio.

```
bool  ChartXYToTimePrice(
   long           chart_id,      // identificador del gráfico
   int            x,             // coordinada X en el gráfico
   int            y,             // coordinada Y en el gráfico
   int&           sub_window,    // número de subventana
   datetime&      time,          // fecha/hora en el gráfico
   double&        price,         // precio en el gráfico
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

x

[in] Coordinada X.

y

[in] Coordinada Y.

sub_window

[out] Variable en la que será escrito el número de la subventana del gráfico. 0 significa la ventana principal del gráfico.

time

[out] Valor de la hora en el gráfico para el cual se recibirá el valor en píxeles en el eje X. El inicio de las coordinadas se encuentra en la esquina superior izquierda de la ventana/subventana.

price

[out] Valor del precio en el gráfico para el cual se recibirá el valor en píxeles en el eje Y. El inicio de las coordinadas se encuentra en la esquina superior izquierda de la ventana/subventana.

Valor devuelto

Devuelve true en caso de la ejecución con éxito, de lo contrario - false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Ejemplo:

```
//--- manejadores de los indicadores
int handle_macd,handle_trix;
```

```
//--- números de subventanas de los indicadores
int win_macd,win_trix;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- creamos la línea horizontal del cursor en cruz
   ObjectCreate(0,"H Line",OBJ_HLINE,0,0,0);
//--- creamos la línea vertical del puntero en cruz
   ObjectCreate(0,"V Line",OBJ_VLINE,0,0,0);
//--- establecemos los valores cero para los manejadores
   handle_macd=0;
   handle_trix=0;
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- eliminamos puntero en cruz
   ObjectDelete(0,"H Line");
   ObjectDelete(0,"V Line");
//--- borramos el comentario
   Comment("");
//--- eliminamos los indicadores
   ChartIndicatorDelete(0,win_macd,ChartIndicatorName(0,win_macd,0));
   ChartIndicatorDelete(0,win_trix,ChartIndicatorName(0,win_trix,0));
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- mostramos los parámetros del evento en el gráfico
   Comment(__FUNCTION__,": id=",id," lparam=",lparam," dparam=",dparam," sparam=",spa
//--- si se trata del evento del cliqueo sobre el gráfico
   if(id==CHARTEVENT_CLICK)
     {
      //--- preparamos las variables
      int      x     =(int)lparam;
      int      y     =(int)dparam;
      datetime dt    =0;
      double   price =0;
      int      window=0;
      //--- convertimos las coordinadas X y Y  en los términos fecha/hora
      if(ChartXYToTimePrice(0,x,y,window,dt,price))
        {
         PrintFormat("Window=%d X=%d  Y=%d  =>  Time=%s  Price=%G",window,x,y,TimeToS
         //--- hacemos la conversión inversa: (X,Y) => (Time,Price)
         if(ChartTimePriceToXY(0,window,dt,price,x,y))
            PrintFormat("Time=%s  Price=%G  =>  X=%d  Y=%d",TimeToString(dt),price,x,
         else
            Print("ChartTimePriceToXY return error code: ",GetLastError());
         //--- establecemos la coordinada de la hora para la línea vertical del punte
         ObjectSetInteger(0,"V Line",OBJPROP_TIME,dt);
```

```
         //--- averiguamos en qué ventana se encuentra la línea horizontal del punter
         int h_line_window=ObjectFind(0,"H Line");
         //--- si la nueva ventana de la línea horizontal no coincide con la ventana
         if(h_line_window!=window)
           {
            //--- eliminamos la línea desde una ventana y la creamos en la otra
            ObjectDelete(0,"H Line");
            ObjectCreate(0,"H Line",OBJ_HLINE,window,dt,price);
           }
         //--- establecemos la coordinada del precio para la línea horizontal del pun
         ObjectSetDouble(0,"H Line",OBJPROP_PRICE,window,price);
         ChartRedraw(0);
        }
      else
         Print("ChartXYToTimePrice return error code: ",GetLastError());
      Print("+------------------------------------------------------------------+");
     }
//--- creamos un par de indicadores en diferentes subventanas para que nuestro ejempl
   if(handle_macd==0 || handle_macd==INVALID_HANDLE)
     {
      handle_macd=iMACD(_Symbol,_Period,12,26,9,PRICE_CLOSE);
      //--- si el manejador se ha creado con éxito
      if(handle_macd!=INVALID_HANDLE)
        {
         //--- averiguamos cuántas ventanas en total se encuentran en el gráfico
         int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
         //--- añadimos el indicador al gráfico en nueva subventana y recordamos el n
         if(ChartIndicatorAdd(0,windows,handle_macd)) win_macd=windows;
        }
     }
//--- creamos el segundo indicador y lo insertamos en el gráfico
   if(handle_trix==0 || handle_trix==INVALID_HANDLE)
     {
      handle_trix=iTriX(_Symbol,_Period,14,PRICE_CLOSE);
      if(handle_trix!=INVALID_HANDLE)
        {
         int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
         if(ChartIndicatorAdd(0,windows,handle_trix)) win_trix=windows;
        }
     }
  }
```

Véase también

ChartTimePriceToXY()

ChartOpen

Abre un gráfico nuevo con un símbolo y período especificados

```
long  ChartOpen(
   string           symbol,     // nombre del símbolo
   ENUM_TIMEFRAMES  period      // período
   );
```

Parámetros

symbol

[in] Símbolo del gráfico. NULL significa el símbolo del gráfico corriente (al que está adjuntado Asesor Experto).

period

[in] Período del gráfico (período de tiempo). Puede adquirir uno de los valores de la enumeración ENUM_TIMEFRAMES. 0 significa el período del gráfico corriente.

Valor devuelto

En caso del éxito la función devuelve el identificador del gráfico. De lo contrario devuelve 0.

Nota

El número máximo posible de gráficos abiertos a la vez en el terminal no puede superar el valor de CHARTS_MAX.

ChartFirst

Devuelve el indicador del primer gráfico del terminal de cliente.

```
long  ChartFirst();
```

Valor devuelto

Identificador del gráfico.

ChartNext

Devuelve el identificador del gráfico que sigue después del gráfico especificado.

```
long  ChartNext(
   long  chart_id      // identificador del gráfico
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 no significa el gráfico corriente. 0 significa "devolver el identificador del primer gráfico".

Valor devuelto

Identificador del gráfico. Si la lista de gráficos se ha terminado, la función devuelve -1.

Ejemplo:

```
//--- variables para los identificadores de gráficos
   long currChart,prevChart=ChartFirst();
   int i=0,limit=100;
   Print("ChartFirst = ",ChartSymbol(prevChart)," ID = ",prevChart);
   while(i<limit)// seguramente no tenemos más de 100 gráficos abiertos
     {
      currChart=ChartNext(prevChart); // a base del anterior obtenemos un gráfico nue
      if(currChart<0) break;          // hemos llegado al final de la lista de gráfic
      Print(i,ChartSymbol(currChart)," ID = ",currChart);
      prevChart=currChart;// vamos a guardar el identificador del gráfico corriente p
      i++;// no olvidemos aumentar el contador
     }
```

ChartClose

Cierra el gráfico especificado.

```
bool  ChartClose(
   long  chart_id=0      // identificador del gráfico
   );
```

Parámetros

chart_id=0

[in] Identificador del gráfico. 0 significa el gráfico actual.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

ChartSymbol

Devuelve el nombre del símbolo del gráfico especificado.

```
string  ChartSymbol(
   long  chart_id=0      // identificador del gráfico
   );
```

Parámetros

chart_id=0

[in] Identificador del gráfico. 0 significa el gráfico actual.

Valor devuelto

Si el gráfico no existe, se devuelve una cadena vacía.

Véase también

ChartSetSymbolPeriod

ChartPeriod

Devuelve el valor del período del gráfico especificado.

```
ENUM_TIMEFRAMES  ChartPeriod(
   long  chart_id=0      // identificador del gráfico
   );
```

Parámetros

chart_id=0

[in] Identificador del gráfico. 0 significa el gráfico actual.

Valor devuelto

Valor del tipo ENUM_TIMEFRAMES. Si el gráfico no existe, se devuelve 0.

ChartRedraw

Activa el redibujo forzado de un gráfico especificado.

```
void  ChartRedraw(
   long  chart_id=0      // identificador del gráfico
   );
```

Parámetros

chart_id=0

[in] Identificador del gráfico. 0 significa el gráfico actual.

Nota

Suele usarse después del cambio de las propiedades de objetos.

Véase también

Objetos gráficos

ChartSetDouble

Establece el valor para la propiedad correspondiente del gráfico especificado. La propiedad del gráfico debe ser del tipo double.

```
bool  ChartSetDouble(
   long     chart_id,      // identificador del gráfico
   int      prop_id,       // identificador de la propiedad
   double   value          // valor
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

prop_id

[in] Identificador de la propiedad del gráfico. Su valor puede ser uno de los valores de la enumeración ENUM_CHART_PROPERTY_DOUBLE (excepto las propiedades read-only).

value

[in] Valor de la propiedad.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

ChartSetInteger

Establece el valor para la propiedad correspondiente del gráfico especificado. La propiedad del gráfico debe ser del tipo datetime, int, color, bool o char.

```
bool  ChartSetInteger(
   long    chart_id,      // identificador del gráfico
   int     prop_id,       // identificador de la propiedad
   long    value          // valor
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

prop_id

[in] Identificador de la propiedad del gráfico. Su valor puede ser uno de los valores de la enumeración ENUM_CHART_PROPERTY_INTEGER (excepto las propiedades read-only).

value

[in] Valor de la propiedad.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

ChartSetString

Establece el valor para la propiedad correspondiente del gráfico especificado. La propiedad del gráfico debe ser string.

```
bool  ChartSetString(
   long  chart_id,          // identificador del gráfico
   int   prop_id,           // identificador de la propiedad
   string   str_value       // valor
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

prop_id

[in] Identificador de la propiedad del gráfico. Su valor puede ser uno de los valores de la enumeración ENUM_CHART_PROPERTY_STRING (salvo read-only).

str_value

[in] Cadena para establecer la propiedad. La longitud de la cadena no puede superar 2045 símbolos (los que sobran, serán recortados).

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

La función ChartSetString puede ser usada para visualizar los comentarios en el gráfico en vez de la función Comment.

Ejemplo:

```
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("Mostramos precios:\nAsk = %G\nBid = %G\nSpread = %d",
                               Ask,Bid,Spread);
   ChartSetString(0,CHART_COMMENT,comment);
  }
```

Véase también

Comment, ChartGetString

ChartGetDouble

Devuelve el valor para la propiedad correspondiente del gráfico especificado. La propiedad del gráfico debe ser del tipo double. Existen 2 variantes de la función.

1. Inmediatamente devuelve el valor de la propiedad.

```
double  ChartGetDouble(
   long  chart_id,          // identificador del gráfico
   int   prop_id,           // identificador de la propiedad
   int   sub_window=0       // número de subventana, si hace falta
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  ChartGetDouble(
   long    chart_id,        // identificador del gráfico
   int     prop_id,         // identificador de la propiedad
   int     sub_window,      // número de subventana
   double& double_var       // aquí recibimos el valor de la propiedad
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

prop_id

[in] Identificador de la propiedad del gráfico. Su valor puede ser uno de los valores de la enumeración ENUM_CHART_PROPERTY_DOUBLE.

sub_window

[in] Número de subventana del gráfico. Para la primera variante por defecto el valor es igual a 0 (ventana principal del gráfico). La mayoría de las propiedades no requiere indicar el número de subventana.

double_var

[out] Variable del tipo double que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo double.

En segundo caso devuelve true, si dicha propiedad estça disponible y su valor ha sido pasado en la variable double_var, de lo contrario devuelve false. Para obtener más detalles sobre el error hay que llamar a la función GetLastError().

Ejemplo:

```
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

Devuelve el valor para la propiedad correspondiente del gráfico especificado. La propiedad del gráfico debe ser del tipo datetime, int o bool. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
long  ChartGetInteger(
   long  chart_id,          // identificador del gráfico
   int   prop_id,           // identificador de la propiedad
   int   sub_window=0       // número de subventana, si hace falta
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  ChartGetInteger(
   long    chart_id,        // identificador del gráfico
   int     prop_id,         // identificador de la propiedad
   int     sub_window,      // número de subventana
   long&   long_var         // aquí recibimos el valor de la propiedad
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

prop_id

[in] Identificador de la propiedad del gráfico. Su valor puede ser uno de los valores de la enumeración ENUM_CHART_PROPERTY_INTEGER.

sub_window

[in] Número de subventana del gráfico. Para la primera variante por defecto el valor es igual a 0 (ventana principal del gráfico). La mayoría de las propiedades no requiere indicar el número de subventana.

long_var

[out] Variable del tipo long que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo long.

En segundo caso devuelve true, si dicha propiedad estça disponible y su valor ha sido pasado en la variable long_var, de lo contrario devuelve false. Para obtener más detalles sobre el error,hay que llamar a la función GetLastError().

Ejemplo:

```
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

Devuelve el valor para la propiedad correspondiente del gráfico especificado. La propiedad del gráfico debe ser del tipo string. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
string  ChartGetString(
   long   chart_id,          // identificador del gráfico
   int    prop_id            // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  ChartGetString(
   long     chart_id,        // identificador del gráfico
   int      prop_id,         // identificador de la propiedad
   string&  string_var       // aquí recibimos el valor de la propiedad
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

prop_id

[in] Identificador de la propiedad del gráfico. Su valor puede ser uno de los valores de la enumeración ENUM_CHART_PROPERTY_STRING.

string_var

[out] Variable del tipo string que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo string.

En segundo caso devuelve true, si dicha propiedad estça disponible y su valor ha sido pasado en la variable string_var, de lo contrario devuelve false. Para obtener más detalles sobre el error hay que llamar a la función GetLastError().

Nota

La función ChartGetString puede ser usada para leer los comentarios se visualizan en el gráfico usando las funciones Comment o ChartSetString.

Ejemplo:

```
void OnStart()
  {
   ChartSetString(0,CHART_COMMENT,"Test comment.\nSecond line.\nThird!");
   ChartRedraw();
   Sleep(1000);
   string comm=ChartGetString(0,CHART_COMMENT);
```

```
   Print(comm);
  }
```

Véase también

Comment, ChartSetString

ChartNavigate

Desplaza el gráfico especificado a una cantidad de barras especificada respecto a la posición del gráfico indicada.

```
bool  ChartNavigate(
   long  chart_id,      // identificador del gráfico
   int   position,      // posición
   int   shift=0        // valor del desplazamiento
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

position

[in] Posición del gráfico respecto a la que va a realizarse el desplazamiento. Su valor puede ser uno de los valores de la enumeración ENUM_CHART_POSITION.

shift=0

[in] Número de barras al que hay que mover el gráfico. El valor positivo supone el desplazamiento a la derecha (al final del gráfico), el valor negativo significa el desplazamiento a la izquierda (al principio del gráfico). El desplazamiento cero tiene sentido cuando navegamos al principio o al final del gráfico.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Ejemplo:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- obtenemos el manejador (handle) del gráfico actual
   long handle=ChartID();
   string comm="";
   if(handle>0) // si ha salido bien, realizaremos ajustes adicionales
     {
      //--- desactivaremos el desplazamiento automático
      ChartSetInteger(handle,CHART_AUTOSCROLL,false);
      //--- establecemos la sangría del borde derecho del gráfico
      ChartSetInteger(handle,CHART_SHIFT,true);
      //--- mostramos en forma de velas
      ChartSetInteger(handle,CHART_MODE,CHART_CANDLES);
      //--- establecemos el modo de visualización de los volúmenes de ticks
      ChartSetInteger(handle,CHART_SHOW_VOLUMES,CHART_VOLUME_TICK);

      //--- preparamos el texto para mostrar en Comment()
```

```
   comm="Desplazamos a 10 barras a la derecha desde el inicio del historial";
   //--- mostramos el comentario
   Comment(comm);
   //--- desplazamos a 10 barras a la derecha desde el inicio del historial
   ChartNavigate(handle,CHART_BEGIN,10);
   //--- obtenemos el número de la primera barra visible en el gráfico (numeración
   long first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
   //--- añadimos el carácter de nueva línea
   comm=comm+"\r\n";
   //--- añadimos comentario
   comm=comm+"La primera barra en el gráfico tiene el número "+IntegerToString(fir
   //--- mostramos el comentario
   Comment(comm);
   //--- esperaremos 5 segundos para que nos de tiempo a ver cómo se desplaza el g
   Sleep(5000);

   //--- completaremos el texto del comentario
   comm=comm+"\r\n"+"Desplazamos a 10 barras a la izquierda desde el borde derecho
   Comment(comm);
   //--- desplazamos a 10 barras a la izquierda desde el borde derecho del gráfico
   ChartNavigate(handle,CHART_END,-10);
   //--- obtenemos el número de la primera barra visible en el gráfico (numeración
   first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
   comm=comm+"\r\n";
   comm=comm+"La primera barra en el gráfico tiene el número "+IntegerToString(fir
   Comment(comm);
   //--- esperaremos 5 segundos para que nos de tiempo a ver cómo se desplaza el g
   Sleep(5000);

   //--- nuevo bloque del desplazamiento del gráfico
   comm=comm+"\r\n"+"Desplazamos a 300 barras a la derecha desde el inicio del his
   Comment(comm);
   //--- desplazamos a 300 barras a la derecha desde el inicio del historial
   ChartNavigate(handle,CHART_BEGIN,300);
   first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
   comm=comm+"\r\n";
   comm=comm+"La primera barra en el gráfico tiene el número "+IntegerToString(fir
   Comment(comm);
   //--- esperaremos 5 segundos para que nos de tiempo a ver cómo se desplaza el g
   Sleep(5000);

   //--- nuevo bloque del desplazamiento del gráfico
   comm=comm+"\r\n"+"Desplazamos a 300 barras a la izquierda desde el borde derech
   Comment(comm);
   //--- desplazamos a 300 barras a la izquierda desde el borde derecho del gráfic
   ChartNavigate(handle,CHART_END,-300);
   first_bar=ChartGetInteger(0,CHART_FIRST_VISIBLE_BAR,0);
   comm=comm+"\r\n";
```

```
      comm=comm+"La primera barra en el gráfico tiene el número "+IntegerToString(fir
      Comment(comm);
     }
  }
```

ChartID

Devuelve el identificador del gráfico corriente.

```
long  ChartID();
```

Valor devuelto

Valor del tipo long.

ChartIndicatorAdd

Añade un indicador con el manejador (handle) especificado a la ventana del gráfico especificada. El indicador y el gráfico tienen que generarse basándose en el mismo símbolo y en el mismo período de tiempo.

```
bool  ChartIndicatorAdd(
   long  chart_id,                  // identificador del gráfico
   int   sub_window                 // número de subventana
   int   indicator_handle           // manejador del indicador
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

sub_window

[in] Número de subventana del gráfico. 0 significa la ventana principal del gráfico. Para añadir un indicador a una ventana nueva, el parámetro debe ser un uno más que el índice de la última ventana existente, es decir debe ser igual a CHART_WINDOWS_TOTAL. Si el valor del parámetro sobrepasa el valor de CHART_WINDOWS_TOTAL, entonces la nueva ventana no se creará y el indicador no será agregado.

indicator_handle

[in] Manejador del indicador.

Valor devuelto

Devuelve true en caso del éxito, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError(). El error 4114 significa que el gráfico y el indicador que se añade se diferencian por el símbolo o por el período de tiempo.

Nota

Si a la ventana principal del gráfico se le añade un indicador que tiene que ser dibujado en una ventana separada (por ejemplo, indicador built-in iMACD o un indicador personalizado con la propiedad especificada #property indicator_separate_window), entonces puede ocurrir que este indicador se quede invisible, aunque va a figurar en la lista de indicadores. Eso quiere decir que la escala de este indicador se diferencia de la escala del gráfico de precios, y los valores del indicador insertado no entran en la extensión visualizada del gráfico de precios. En este caso GetLastError() va a devolver el código nulo que indica en la ausencia de errores. Se puede observar el valor de este indicador "invisible" en la "Ventana de Datos", así como obtenerlo desde otros programas MQL5.

Ejemplo:

```
#property description "El EA para demostrar el trabajo con la función ChartIndicatorA
#property description "Después de arrancarlo en el gráfico (y recibir el error en el
#property description "las propiedades del EA y establezca los parámetros correctos d
#property description "El indicador MACD sera agregado al gráfico."

//--- input parameters
input string          symbol="AUDUSD";     // nombre del símbolo
input ENUM_TIMEFRAMES period=PERIOD_M12;   // período de tiempo
input int    fast_ema_period=12;           // período de media rápida MACD
input int    slow_ema_period=26;           // período de media lenta MACD
input int      signal_period=9;            // período del promedio de diferencia
input ENUM_APPLIED_PRICE apr=PRICE_CLOSE; // tipo del precio para el cálculo MACD

int indicator_handle=INVALID_HANDLE;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//---
   indicator_handle=iMACD(symbol,period,fast_ema_period,slow_ema_period,signal_period
//--- intentamos insertar el indicador en el gráfico
   if(!AddIndicator())
     {
      //--- la función AddIndicator() ha negado a insertar el indicador en el gráfico
      int answer=MessageBox("¿Desea intentar agregar MACD al gráfico de cualquier mod
                            "El símbolo y/o el período de tiempo no han sido establec
                            MB_YESNO // se mostrarán los botones de selección "Yes" y
                            );
      //--- si el usuario aun así insiste en el uso incorrecto de ChartIndicatorAdd()
      if(answer==IDYES)
        {
         //--- primero vamos a reflejarlo en el Diario
         PrintFormat("¡Atención! %s: Intentemos agregar el indicador MACD(%s/%s) al g
                     __FUNCTION__,symbol,EnumToString(period),_Symbol,EnumToString(_P
         //--- obtenemos el número de una nueva subventana en la que tratamos de inse
         int subwindow=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
         //--- ahora hagamos el intento condenado al error
         if(!ChartIndicatorAdd(0,subwindow,indicator_handle))
            PrintFormat("Fallo al agergar el indicador MACD a la ventana %d del gráfi
                        subwindow,GetLastError());
        }
     }
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
// el EA no hace nada
  }
//+------------------------------------------------------------------+
//| La función para comprobar y añadir el indicador al gráfico       |
//+------------------------------------------------------------------+
bool AddIndicator()
  {
//--- mostramos el mensaje
   string message;
//--- comprobamos si el símbolo del indicador coincide con el símbolo del gráfico
```

```
   if(symbol!=_Symbol)
     {
      message="Demostración del usu de la función Demo_ChartIndicatorAdd():";
      message=message+"\r\n";
      message=message+"No se puede agregar al gráfico un indicador calculado para otr
      message=message+"\r\n";
      message=message+"Especifique el símbolo del gráfico en las propiedades del EA -
      Alert(message);
      //--- salida anticipada, no vamos a añadir el indicador al gráfico
      return false;
     }
//--- comprobaremos si el periodo del indicador coincide con el período del gráfico
   if(period!=_Period)
     {
      message="No se puede agregar al gráfico un indicador calculado para otro períod
      message=message+"\r\n";
      message=message+"Especifique el período del gráfico en las propiedades del EA -
      Alert(message);
      //--- salida anticipada, no vamos a añadir el indicador al gráfico
      return false;
     }
//--- todas las pruebas pasadas, el símbolo y el período del indicador corresponden a
   if(indicator_handle==INVALID_HANDLE)
     {
      Print(__FUNCTION__,"  Creamos el indicador MACD");
      indicator_handle=iMACD(symbol,period,fast_ema_period,slow_ema_period,signal_per
      if(indicator_handle==INVALID_HANDLE)
        {
         Print("Generación del indicador MACD fallida. Código del error ",GetLastErro
        }
     }
//--- reseteamos el código del error
   ResetLastError();
//--- agregamos el indicador al gráfico
   Print(__FUNCTION__,"  Agregamos el indicador MACD al gráfico");
   Print("MACD construida sobre ",symbol,"/",EnumToString(period));
//--- obtenemos el número de nueva subventana a la que vamos a agregar el indicador M
   int subwindow=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   PrintFormat("Agregamos el indicador MACD a la ventana %d del gráfico",subwindow);
   if(!ChartIndicatorAdd(0,subwindow,indicator_handle))
     {
      PrintFormat("Fallo al agergar el indicador MACD a la ventana %d del gráfico. Có
                  subwindow,GetLastError());
     }
//--- el indicador ha sido insertado al gráfico con éxito
   return(true);
  }
```

Véase también

ChartIndicatorDelete(), ChartIndicatorName(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate()

ChartIndicatorDelete

Quita el indicador con el nombre especificado de la ventana del gráfico especificada.

```
bool  ChartIndicatorDelete(
   long           chart_id,              // identificador del gráfico
   int            sub_window             // número de subventana
   const string   indicator_shortname    // nombre breve del indicador
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

sub_window

[in] Número de subventana del gráfico. 0 significa la ventana principal del gráfico.

const indicator_shortname

[in] Nombre breve del indicador que se establece en la propiedad INDICATOR_SHORTNAME por la función IndicatorSetString(). Para obtener el nombre breve del indicador se usa la función ChartIndicatorName().

Valor devuelto

La función devuelve true si el indicador se elimina con éxito, de lo contrario se devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Si en la subventana del gráfico especificada hay varios indicadores con el mismo nombre breve, se elimina el que va primero.

Si en el mismo gráfico están construidos otros indicadores a base de los valores del indicador que se elimina, éstos también serán eliminados.

No se debe confundir el nombre breve del indicador con el nombre del archivo que se indica durante la creación del indicador por las funciones iCustom() y IndicatorCreate(). Si el nombre breve del indicador no está establecido de una manera explícita, entonces durante la compilación ahí se indica el nombre del archivo que contiene el código fuente del indicador.

Si un indicador se quita del gráfico, esto no significa que su parte de cálculo también será eliminada de la memoria del terminal. Para liberar el manejador del indicador, hay que usar la funciónIndicatorRelease().

Se debe formar correctamente el nombre breve del indicador. Este nombre se escribe en la propiedad INDICATOR_SHORTNAME mediante la función IndicatorSetString(). Es recomendable que el nombre breve contenga los valores de los parámetros de entrada del indicador, puesto que en la función ChartIndicatorDelete() la identificación del indicador que se quita del gráfico se realiza precisamente por el nombre breve.

Ejemplo de eliminación de un indicador tras el fallo de inicialización:

```
//+------------------------------------------------------------------+
//|                                    Demo_ChartIndicatorDelete.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   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;
//--- vinculamos el array HistogramBuffer al búfer de indicador
   SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- generamos el nombre breve del indicador a base de los parámetros de entrada
   shortname=StringFormat("Demo_ChartIndicatorDelete(%d,%d,%d)",
                          first_param,second_param,third_param);
   IndicatorSetString(INDICATOR_SHORTNAME,shortname);
//--- si está establecida la finalización forzosa del indicador, devolvemos el valor
   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[])
  {
//--- posición inicial para trabajar en el ciclo
   int start=prev_calculated-1;
   if(start<0) start=0;
//--- llenamos el búfer de indicador con los valores
   for(int i=start;i<rates_total;i++)
     {
```

```
      HistogramBuffer[i]=close[i];
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| manejador de eventos Deinit                                      |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
   PrintFormat("%s: Código de la causa de deinicialización=%d",__FUNCTION__,reason);
   if(reason==REASON_INITFAILED)
     {
      PrintFormat("El indicador con el nombre breve %s (archivo %s) elimina a sí mism
      int window=ChartWindowFind();
      bool res=ChartIndicatorDelete(0,window,shortname);
      //--- analizaremos el resultado de la llamada a ChartIndicatorDelete()
      if(!res)
        {
         PrintFormat("Fallo al eliminar el indicador %s desde la ventana #%d. Código
                     shortname,window,GetLastError());
        }
     }
  }
```

Véase también

ChartIndicatorAdd(), ChartIndicatorName(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString()

ChartIndicatorGet

Devuelve el manejador del indicador con el nombre corto especificado en la ventana del gráfico especificada.

```
int  ChartIndicatorGet(
   long           chart_id,              // identificador del gráfico
   int            sub_window             // número de subventana
   const string   indicator_shortname    // nombre corto del indicador
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

sub_window

[in] Número de subventana del gráfico. 0 significa la ventana principal del gráfico.

const indicator_shortname

[in] El nombre corto del indicador que se establece en la propiedad INDICATOR_SHORTNAME a través de la función IndicatorSetString(). Para obtener el nombre corto del indicador, utilice la función ChartIndicatorName().

Valor devuelto

Devuelve el manejador del indicador en caso de la ejecución con éxito, de lo contrario INVALID_HANDLE. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

A la hora de crear un indicador, hace falta formar de forma correcta su nombre corto que a través de la función IndicatorSetString() se escribe en la propiedad INDICATOR_SHORTNAME. Se recomienda que el nombre corto contenga los valores de los parámetros de entrada del indicador, ya que la identificación del indicador en la función ChartIndicatorGet() se realiza precisamente por el nombre corto.

Otro modo de identificar un indicador es recibir la lista de sus parámetros para el manejador establecido a través de la función IndicatorParameters() y luego realizar el análisis de valores recibidos.

Ejemplo:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   //--- número de ventanas en el gráfico (siempre hay por lo menos una ventana princ
   int windows=(int)ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   //--- repasamos las ventanas
   for(int w=0;w<windows;w++)
     {
      //--- cuántos indicadores hay en esta ventana/subventana
      int total=ChartIndicatorsTotal(0,w);
      //--- repasamos todos los indicadores en la ventana
      for(int i=0;i<total;i++)
        {
         //--- obtenemos el nombre corto del indicador
         string name=ChartIndicatorName(0,w,i);
         //--- obtenemos el manejador del indicador
         int handle=ChartIndicatorGet(0,w,name);
         //--- mostramos en el diario
         PrintFormat("Window=%d,  index=%d,  Name=%s,  handle=%d",w,i,name,handle);
        }
     }
  }
```

Véase también

ChartIndicatorAdd(), ChartIndicatorName(), ChartIndicatorsTotal(), IndicatorParameters()

ChartIndicatorName

Devuelve el nombre breve del indicador según su número en la lista de indicadores en la determinada ventana del gráfico.

```
string  ChartIndicatorName(
   long  chart_id,      // identificador del gráfico
   int   sub_window     // número de subventana
   int   index          // índice del indicador en la lista de indicadores agregados
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

sub_window

[in] Número de subventana del gráfico. 0 significa la ventana principal del gráfico.

index

[in] Índice del indicador en la lista de indicadores. La numeración de indicadores se empieza desde cero, es decir el primer indicador de la lista tiene el índice cero. El número de indicadores en la lista se obtiene a través de la función ChartIndicatorsTotal().

Valor devuelto

La función devuelve el nombre breve del indicador que se establece en la propiedad INDICATOR_SHORTNAME por la función IndicatorSetString(). Para obtener el nombre breve, se puede usar la función ChartIndicatorName(). Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

No se debe confundir el nombre breve del indicador con el nombre del archivo que se indica durante la creación del indicador por las funciones iCustom() y IndicatorCreate(). Si el nombre breve del indicador no está establecido de una manera explícita, entonces durante la compilación ahí se indica el nombre del archivo que contiene el código fuente del indicador.

Si un indicador se quita del gráfico, esto no significa que su parte de cálculo también será eliminada de la memoria del terminal. Para liberar el manejador del indicador, hay que usar la funciónIndicatorRelease().

Se debe formar correctamente el nombre breve del indicador. Este nombre se escribe en la propiedad INDICATOR_SHORTNAME mediante la función IndicatorSetString(). Es recomendable que el nombre breve contenga los valores de los parámetros de entrada del indicador, puesto que en la función ChartIndicatorDelete() la identificación del indicador que se quita del gráfico se realiza precisamente por el nombre breve.

Véase también

ChartIndicatorAdd(), ChartIndicatorDelete(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString()

ChartIndicatorsTotal

Devuelve el número de todos los indicadores vinculados con la determinada ventana del gráfico.

```
int  ChartIndicatorsTotal(
   long  chart_id,       // identificador del gráfico
   int   sub_window      // número de subventana
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

sub_window

[in] Número de subventana del gráfico. 0 significa la ventana principal del gráfico.

Valor devuelto

Número de indicadores en la ventana del gráfico especificada. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

La función sirve para organizar el repaso de todos los indicadores enlazados con este gráfico. El número de todas las ventanas del gráfico se puede obtener desde la propiedad CHART_WINDOWS_TOTAL mediante la función ChartGetInteger().

Véase también

ChartIndicatorAdd(), ChartIndicatorDelete(), ChartIndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString()

ChartWindowOnDropped

Devuelve el número de la subventana del gráfico a la que el Asesor Experto o script ha sido arrastrado con el ratón. 0 significa la ventana principal del gráfico.

```
int  ChartWindowOnDropped();
```

Valor devuelto

Valor del tipo int.

Ejemplo:

```
   int myWindow=ChartWindowOnDropped();
   int windowsTotal=ChartGetInteger(0,CHART_WINDOWS_TOTAL);
   Print("Script iniciado en la ventana #"+myWindow+
         ". El total de ventanas en el gráfico "+ChartSymbol()+": ",windowsTotal);
```

Véase también

ChartPriceOnDropped, ChartTimeOnDropped, ChartXOnDropped, ChartYOnDropped

ChartPriceOnDropped

Devuelve la coordinada de precios que corresponde al punto al que el Asesor Experto o script ha sido arrastrado con el ratón.

```
double  ChartPriceOnDropped();
```

Valor devuelto

Valor del tipo double.

Ejemplo:

```
   double p=ChartPriceOnDropped();
   Print("ChartPriceOnDropped() = ",p);
```

Véase también

ChartXOnDropped, ChartYOnDropped

ChartTimeOnDropped

Devuelve la coordinada de tiempo que corresponde al punto al que el Asesor Experto o script ha sido arrastrado con el ratón.

```
datetime  ChartTimeOnDropped();
```

Valor devuelto

Valor del tipo datetime.

Ejemplo:

```
   datetime t=ChartTimeOnDropped();
   Print("Script wasdropped on the "+t);
```

Véase también

ChartXOnDropped, ChartYOnDropped

ChartXOnDropped

Devuelve la coordinada del eje de X que corresponde al punto al que el Asesor Experto o script ha sido arrastrado con el ratón.

```
int  ChartXOnDropped();
```

Valor devuelto

Valor de coordinada X.

Nota

El eje X va de izquierda a derecha.

Ejemplo:

```
   int X=ChartXOnDropped();
   int Y=ChartYOnDropped();
   Print("(X,Y) = ("+X+","+Y+")");
```

Véase también

ChartWindowOnDropped, ChartPriceOnDropped, ChartTimeOnDropped

ChartYOnDropped

Devuelve la coordinada del eje de Y que corresponde al punto al que el Asesor Experto o script ha sido arrastrado con el ratón.

```
int  ChartYOnDropped();
```

Valor devuelto

Valor de coordinada Y.

Nota

El eje Y va desde arriba hacia abajo.

Véase también

ChartWindowOnDropped, ChartPriceOnDropped, ChartTimeOnDropped

ChartSetSymbolPeriod

Cambia el valor del símbolo y período del gráfico especificado. Esta función es asíncrona, es decir, envía un comando y no espera la finalización de su ejecución.

```
bool  ChartSetSymbolPeriod(
   long             chart_id,      // identificador del gráfico
   string           symbol,        // nombre del símbolo
   ENUM_TIMEFRAMES  period         // período
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

symbol

[in] Símbolo del gráfico. NULL significa el símbolo del gráfico corriente al que está adjuntado Asesor Experto.

period

[in] Período del gráfico (período de tiempo). Puede adquirir uno de los valores de la enumeración ENUM_TIMEFRAMES. 0 significa el período del gráfico corriente.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

El cambio del símbolo/período provoca la reinicialización del Asesor Experto adjuntado al gráfico correspondiente.

Véase también

ChartSymbol, ChartPeriod

ChartScreenShot

Esta función proporciona la captura de pantalla del gráfico especificado en su estado actual en formato gif.

```
bool  ChartScreenShot(
   long             chart_id,                  // identificador del gráfico
   string           filename,                  // nombre del archivo
   int              width,                     // ancho
   int              height,                    // alto
   ENUM_ALIGN_MODE  align_mode=ALIGN_RIGHT     // tipo de alineación
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

filename

[in] Nombre del archivo de screenshot. No puede ser más de 63 símbolos. El archivo se coloca en el directorio \Files.

width

[in] Ancho de screenshot en píxeles.

height

[in] Alto de screenshot en píxeles.

align_mode=ALIGN_RIGHT

[in] Modo output de un screenshot estrecho. Valor de enumeración ENUM_ALIGN_MODE. ALIGN_RIGHT significa la alineación por el margen derecho (output desde el final). ALIGN_LEFT significa la alineación por la izquierda.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

Si hace falta tomar un screenshot del gráfico desde una posición concreta, primero hay que posicionar el gráfico usando la función ChartNavigate(). Si el tamaño horizontal del screenshot es menos que la ventana del gráfico, se toma la parte derecha o izquierda de la ventana del gráfico dependiendo del parámetro align_mode.

Ejemplo:

```
#property description "El Asesor Experto muestra cómo se crea una serie de screenshot
#property description "utilizando la función ChartScreenShot(). Para que sea más cómo
#property description "se visualiza en el gráfico. Las macros determinan el alto y el

#define        WIDTH  800     // ancho de la imagen para llamar a ChartScreenShot()
#define        HEIGHT 600     // alto de la imagen para llamar a ChartScreenShot()
```

```
//--- input parameters
input int      pictures=5;    // número de imágenes en la serie
int            mode=-1;       // -1 significa el desplazamiento hacia el lado derecho
int            bars_shift=300;// número de barras durante el desplazamiento del gráfi
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- desactivamos el desplazamiento automático del gráfico
   ChartSetInteger(0,CHART_AUTOSCROLL,false);
//--- establecemos la sangría del borde derecho del gráfico
   ChartSetInteger(0,CHART_SHIFT,true);
//--- mostramos el gráfico como una secuencia de velas japonesas
   ChartSetInteger(0,CHART_MODE,CHART_CANDLES);
//---
   Print("La preparación del EA para el trabajo está finalizada");
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//---

  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- mostrar el nombre de la función, hora de la llamada y el identificador del even
   Print(__FUNCTION__,TimeCurrent()," id=",id," mode=",mode);
//--- procesamiento del evento CHARTEVENT_CLICK ("Clic del ratón en el gráfico")
   if(id==CHARTEVENT_CLICK)
     {
      //--- desplazamiento inicial del borde del gráfico
      int pos=0;
      //--- modo de trabajo con el borde izquierdo del gráfico
      if(mode>0)
        {
         //--- desplazamos el gráfico hacia el borde izquierdo
         ChartNavigate(0,CHART_BEGIN,pos);
         for(int i=0;i<pictures;i++)
           {
```

```
            //--- preparamos el texto a mostrar en el gráfico y el nombre para el arc
            string name="ChartScreenShot"+"CHART_BEGIN"+string(pos)+".gif";
            //--- mostramos el nombre en el gráfico en forma del comentario
            Comment(name);
            //--- guardamos el screenshot del gráfico en la carpeta directorio_del_te
            if(ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_LEFT))
               Print("Hemos guardado el screenshot ",name);
            //---
            pos+=bars_shift;
            //--- dejamos al usuario tiempo para que mire una nueva área del gráfico
            Sleep(3000);
            //--- desplazamos el gráfico a bars_shift a la derecha de su posición act
            ChartNavigate(0,CHART_CURRENT_POS,bars_shift);
           }
         //--- cambio del modo al opuesto
         mode*=-1;
        }
      else // modo de trabajo con el borde derecho del gráfico
        {
         //--- desplazamos el gráfico hacia el borde derecho
         ChartNavigate(0,CHART_END,pos);
         for(int i=0;i<pictures;i++)
           {
            //--- preparamos el texto a mostrar en el gráfico y el nombre para el arc
            string name="ChartScreenShot"+"CHART_END"+string(pos)+".gif";
            //--- mostramos el nombre en el gráfico en forma del comentario
            Comment(name);
            //--- guardamos el screenshot del gráfico en la carpeta directorio_del_te
            if(ChartScreenShot(0,name,WIDTH,HEIGHT,ALIGN_RIGHT))
               Print("Hemos guardado el screenshot ",name);
            //---
            pos+=bars_shift;
            //--- dejamos al usuario tiempo para que mire una nueva área del gráfico
            Sleep(3000);
            //--- desplazamos el gráfico a bars_shift a la derecha de su posición act
            ChartNavigate(0,CHART_CURRENT_POS,-bars_shift);
           }
         //--- cambio del modo al opuesto
         mode*=-1;
        }
     }  // fin del procesamiento del evento CHARTEVENT_CLICK
//--- fin del manejador OnChartEvent()
  }
```

Véase también

ChartNavigate(), Recursos

Funciones comerciales

Es el grupo de funciones que sirven para gestionar la actividad comercial.

Estas funciones pueden usarse en los Asesores Expertos y scripts. Las funciones comerciales pueden ser invocadas sólo si en las propiedades del Asesor Experto o script correspondiente está activada la opción "Permitir comerciar al Asesor Experto".

Función	Acción
OrderCalcMargin	Calcula el margen requerido para el tipo de orden especificado en la moneda de depósito de la cuenta
OrderCalcProfit	Calcula el beneficio basado en los parámetros pasados en la moneda de depósito de la cuenta
OrderCheck	Comprueba si la cuenta dispone de fondos suficientes para ejecutar la operación comercial requerida
OrderSend	Comprueba si hay suficientes fondos para ejecutar la operación comercial especificada.
OrderSendAsync	Envía de modo asincrónico las solicitudes comerciales sin esperar la respuesta por parte del servidor de trading
PositionsTotal	Devuelve el número de posiciones abiertas
PositionGetSymbol	Devuelve el símbolo de una posición correspondiente abierta
PositionSelect	Elige una posición abierta para el futuro trabajo con ella
PositionGetDouble	Devuelve la propiedad solicitada de una posición abierta (double)
PositionGetInteger	Devuelve la propiedad solicitada de una posición abierta (datetime o int)
PositionGetString	Devuelve la propiedad solicitada de una posición abierta (string)
OrdersTotal	Devuelve el número de órdenes
OrderGetTicket	Devuelve el ticket de una orden correspondiente
OrderSelect	Elige una orden para el futuro trabajo con ella
OrderGetDouble	Devuelve la propiedad solicitada de una orden (double)
OrderGetInteger	Devuelve la propiedad solicitada de una orden (datetime o int)

OrderGetString	Devuelve la propiedad solicitada de una orden (string)
HistorySelect	Solicita el historial de transacciones y órdenes del período especificado de la hora del servidor
HistoryOrderSelect	Elige en el historial una orden para el futuro trabajo con ella
HistorySelectByPosition	Solicita el historial de transacciones y órdenes con el identificador de posición especificado
HistoryOrdersTotal	Devuelve el número de órdenes en el historial
HistoryOrderGetTicket	Devuelve el ticket de una orden correspondiente en el historial
HistoryOrderGetDouble	Devuelve la propiedad solicitada de una orden en el historial (double)
HistoryOrderGetInteger	Devuelve la propiedad solicitada de una orden en el historial (datetime o int)
HistoryOrderGetString	Devuelve la propiedad solicitada de una orden en el historial (string)
HistoryDealSelect	Elige en el historial una transacción para dirigirse a ella en el futuro mediante las funciones correspondientes
HistoryDealsTotal	Devuelve el número de transacciones en el historial
HistoryDealGetTicket	Elige una transacción a procesar y devuelve el ticket de transacción en el historial
HistoryDealGetDouble	Devuelve la propiedad solicitada de una transacción en el historial (double)
HistoryDealGetInteger	Devuelve la propiedad solicitada de una transacción en el historial (datetime o int)
HistoryDealGetString	Devuelve la propiedad solicitada de una transacción en el historial (string)

OrderCalcMargin

Calcula el margen requerido para el tipo de orden especificado , en la cuenta actual, en el entorno del mercado actual y sin tener en cuenta las órdenes pendientes actuales y posiciones abiertas. Permite evaluar el margen requerido para la operación comercial planeada. El valor se devuelve en la moneda de la cuenta.

```
bool  OrderCalcMargin(
   ENUM_ORDER_TYPE       action,          // tipo de orden
   string                symbol,          // nombre del símbolo
   double                volume,          // volumen
   double                price,           // precio de apertura
   double&               margin           // variable para la obtención del valor de
   );
```

Parámetros

action

[in] Tipo de orden, puede ser uno de los valores de la enumeración ENUM_ORDER_TYPE.

symbol

[in] Nombre del símbolo.

volume

[in] Volumen de la operación comercial a evaluar.

price

[in] Precio de apertura.

margin

[out] Variable en la que se recibirá el valor del margen requerido si la función se ejecuta con éxito. El cálculo se realiza sin tomar en consideración las órdenes pendientes ni las posiciones abiertas que pudieran haber en la cuenta actual. El valor del margen depende de muchos factores y puede ser diferente en diferentes entornos del mercado.

Valor devuelto

Devuelve true en caso de éxito. De lo contrario, la función devuelve false. Para obtener la información acerca del error, se debe usar la función GetLastError().

Véase también

OrderSend(), Propiedades de órdenes, Tipos de operaciones comerciales

OrderCalcProfit

Esta función calcula el beneficio para la cuenta actual en las condiciones actuales del mercado a base de los parámetros pasados. La función se utiliza para la evaluación previa de los resultados de una operación comercial. El valor se devuelve en la moneda de la cuenta.

```
bool  OrderCalcProfit(
   ENUM_ORDER_TYPE       action,          // tipo de orden (ORDER_TYPE_BUY o ORDER_T
   string                symbol,          // nombre del símbolo
   double                volume,          // volumen
   double                price_open,      // precio de apertura
   double                price_close,     // precio de cierre
   double&               profit           // variable donde se recibe el valor del be
   );
```

Parámetros

action

[in] Tipo de orden, puede ser uno de los valores de la enumeración ENUM_ORDER_TYPE: ORDER_TYPE_BUY o ORDER_TYPE_SELL.

symbol

[in] Nombre del símbolo.

volume

[in] Volumen de la operación comercial.

price_open

[in] Precio de apertura.

price_close

[in] Precio de cierre.

profit

[out] Variable en la que se recibirá el valor del beneficio en caso de que la función se ejecute con éxito. El valor del beneficio estimado depende de muchos factores y puede variar en diferentes entornos del mercado.

Valor devuelto

Devuelve true en caso de éxito. De lo contrario, la función devuelve false. Si se indica el tipo de orden no admisible, la función devolverá false. Para obtener la información sobre el error ocurrido, hay que llamar a la función GetLastError().

Véase también

OrderSend(), Propiedades de órdenes, Tipos de operaciones comerciales

OrderCheck

La función OrderCheck() comprueba si la cuenta dispone de fondos suficientes para ejecutar la operación comercial requerida. Los resultados de la comprobación se colocan en los campos de la estructura MqlTradeCheckResult.

```
bool  OrderCheck(
   MqlTradeRequest&       request,      // estructura de la solicitud comercial
   MqlTradeCheckResult&   result        // estructura de la respuesta
   );
```

Parámetros

request

[in] Puntero a una estructura del tipo MqlTradeRequest que describe la acción comercial requerida.

result

[in,out] Puntero a una estructura del tipo MqlTradeCheckResult en la que se colocarán los resultados de la comprobación.

Valor devuelto

Si los fondos de la cuenta no son suficientes para la operación especificada, o los parámetros son incorrectos, la función devuelve false. Si la comprobación básica de las estructuras (comprobación de punteros) se ha realizado con éxito, la función devuelve true (**lo que no significa que la operación comercial que se solicita vaya a ser ejecutada con éxito**). Para una descripción más detallada de los resultados de la ejecución de esta función, hay que analizar los campos de la estructura *result*.

Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Véase también

OrderSend(), Tipos de operaciones comerciales, Estructura de solicitud comercial, Estructura de comprobación de solicitud comercial, Estructura de resultado de solicitud comercial

OrderSend

La función OrderSend() está destinada para realizar la actividad comercial dentro de los márgenes de MQL5.

```
bool  OrderSend(
   MqlTradeRequest&  request      // estructura de petición
   MqlTradeResult&   result       // estructura de respuesta
   );
```

Parámetros

request

[in] Puntero a una estructura del tipo MqlTradeRequest que describe la acción comercial del cliente.

result

[in,out] Puntero a una estructura del tipo MqlTradeResult que describe el resultado de una operación comercial en caso de llevarla a cabo con éxito (se devuelve true).

Valor devuelto

En caso de comprobar las estructuras (comprobación de punteros) con éxito, la función devuelve true, **pero eso no significa que la operación comercial que se solicita vaya a ser ejecutada con éxito**. Para una descripción más detallada de los resultados de la ejecución de esta función, hay que analizar los campos de la estructura *result*.

Nota

Las solicitudes comerciales pasan por varias fases de comprobación en el servidor comercial. En primer lugar se comprueba si todos los campos necesarios del parámetro *request* están rellenados correctamente. Si no hay errores, el servidor acepta la solicitud para su procesamiento. Si la orden se acepta con éxito, la función OrderSend() devuelve el valor true.

Se recomienda comprobar personalmente la solicitud antes de enviarla al servidor comercial. Para eso sirve la función OrderCheck() que no sólo comprueba si hay fondos suficientes en la cuenta para la ejecución de la operación comercial, sino también devuelve muchos otros parámetros útiles como resultado de comprobación de la solicitud comercial:

- código de retorno que avisa sobre un error en la solicitud que se comprueba;
- valor del balance de la cuenta que se queda tras la ejecución de la operación comercial;
- valor de fondos propios que se obtiene tras la ejecución de la operación comercial;
- valor del beneficio flotante que se obtiene tras la ejecución de la operación comercial;
- margen necesario para la operación comercial;
- fondos propios que se quedan disponibles después de realizar la operación comercial;
- nivel del margen que va a establecerse después de realizar la operación comercial;
- comentario sobre el código de respuesta, descripción del error en su caso.

A la hora de colocar una orden de mercado hay que tener en cuenta que la ejecución satisfactoria de la función OrderSend() no siempre significa que la transacción se vaya a completar con éxito. Por esta razón en la estructura del resultado *result* hay que comprobar el valor *retcode* que contiene el código de respuesta del servidor comercial, y los valores de los campos *deal* o *order*, dependiendo del tipo de operación.

Cada orden aceptada se almacena en el servidor comercial esperando su procesamiento hasta que se de una de las condiciones adecuadas para su ejecución:

- expiración del plazo de vigencia,
- aparición de una solicitud opuesta,
- ejecución de la orden tras la recepción del precio de ejecución,
- aparición de la solicitud de cancelación de la orden.

Durante el procesamiento de la orden el servidor comercial envía un mensaje al terminal sobre el evento Trade, que se puede procesar mediante la función OnTrade().

En el servidor el resultado de ejecución de la solicitud comercial que ha sido enviada con la función OrderSend() se puede seguir a través del manejador OnTradeTransaction. Hay que tener en cuenta que durante la ejecución de una solicitud comercial el manejador OnTradeTransaction será invocado varias veces.

Por ejemplo, al enviar una orden de compra, ésta se tramita, para la cuenta se crea una orden de compra correspondiente, se realiza la ejecución de la orden, su eliminación de la lista de las abiertas, se agrega al historial de órdenes, luego la operación correspondiente se agrega al historial, y se crea una posición nueva. La función OnTradeTransaction será llamada para cada uno de estos eventos.

Ejemplo:

```
//--- valores para ORDER_MAGIC
input long order_magic=55555;
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- vamos a comprobar que se trata de una cuenta de demostración
   if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
     {
      Alert("Operación del script en una cuenta real está prohibida!");
      return;
     }
//--- colocamos o eliminamos una orden
   if(GetOrdersTotalByMagic(order_magic)==0)
     {
      //--- no hay órdenes corrientes - colocamos una orden
      uint res=SendRandomPendingOrder(order_magic);
      Print("Código de devolución del servidor comercial ",res);
     }
   else // hay órdenes - eliminamos órdenes
     {
      DeleteAllOrdersByMagic(order_magic);
     }
//---
  }
//+------------------------------------------------------------------+
//|  recibir la cantidad actual de órdenes con ORDER_MAGIC especificada |
```

```
//+------------------------------------------------------------------+
int GetOrdersTotalByMagic(long const magic_number)
  {
   ulong order_ticket;
   int total=0;
//--- repasamos todas las órdenes pendientes
   for(int i=0;i<OrdersTotal();i++)
      if((order_ticket=OrderGetTicket(i))>0)
         if(magic_number==OrderGetInteger(ORDER_MAGIC)) total++;
//---
   return(total);
  }
//+------------------------------------------------------------------+
//| elimina todas las órdenes pendientes con ORDER_MAGIC especifcada |
//+------------------------------------------------------------------+
void DeleteAllOrdersByMagic(long const magic_number)
  {
   ulong order_ticket;
//--- repasamos todas las órdenes pendientes
   for(int i=0;i<OrdersTotal();i++)
      if((order_ticket=OrderGetTicket(i))>0)
         //--- orden con ORDER_MAGIC apropiada
         if(magic_number==OrderGetInteger(ORDER_MAGIC))
           {
            MqlTradeResult result={0};
            MqlTradeRequest request={0};
            request.order=order_ticket;
            request.action=TRADE_ACTION_REMOVE;
            OrderSend(request,result);
            //--- apuntamos en el log la respuesta del servidor
            Print(__FUNCTION__,": ",result.comment," código de respuesta ",result.ret
           }
//---
  }
//+------------------------------------------------------------------+
//|  establecer una orden pendiente de una manera aleatoria          |
//+------------------------------------------------------------------+
uint SendRandomPendingOrder(long const magic_number)
  {
//--- preparamos la solicitud
   MqlTradeRequest request={0};
   request.action=TRADE_ACTION_PENDING;          // definir una orden pendiente
   request.magic=magic_number;                   // ORDER_MAGIC
   request.symbol=_Symbol;                       // instrumento
   request.volume=0.1;                           // volumen de 0.1 lote
   request.sl=0;                                 // Stop Loss sin especificar
   request.tp=0;                                 // Take Profit sin especificar
//--- vamos a formar el tipo de orden
```

```
   request.type=GetRandomType();                  // tipo de orden
//---vamos a formar el precio para una orden pendiente
   request.price=GetRandomPrice(request.type);  // precio de apertura
//--- enviamos la orden comercial
   MqlTradeResult result={0};
   OrderSend(request,result);
//--- introducimos la respuesta del servidor en el log
   Print(__FUNCTION__,":",result.comment);
   if(result.retcode==10016) Print(result.bid,result.ask,result.price);
//--- devolvemos el código de retorno del servidor comercial
   return result.retcode;
  }
//+------------------------------------------------------------------+
//| recibir el tipo de una orden pendiente de una manera aleatoria   |
//+------------------------------------------------------------------+
ENUM_ORDER_TYPE GetRandomType()
  {
   int t=MathRand()%4;
//---   0<=t<4
   switch(t)
     {
      case(0):return(ORDER_TYPE_BUY_LIMIT);
      case(1):return(ORDER_TYPE_SELL_LIMIT);
      case(2):return(ORDER_TYPE_BUY_STOP);
      case(3):return(ORDER_TYPE_SELL_STOP);
     }
//--- valor incorrecto
   return(WRONG_VALUE);
  }
//+------------------------------------------------------------------+
//|  recibir el precio de una manera aleatoria                       |
//+------------------------------------------------------------------+
double GetRandomPrice(ENUM_ORDER_TYPE type)
  {
   int t=(int)type;
//--- nivel de stops para el símbolo
   int distance=(int)SymbolInfoInteger(_Symbol,SYMBOL_TRADE_STOPS_LEVEL);
//--- recibimos los datos del último tick
   MqlTick last_tick={0};
   SymbolInfoTick(_Symbol,last_tick);
//--- vamos a calcular el precio de acuerdo con el tipo
   double price;
   if(t==2 || t==5) // ORDER_TYPE_BUY_LIMIT o ORDER_TYPE_SELL_STOP
     {
      price=last_tick.bid; // partimos del precio Bid
      price=price-(distance+(MathRand()%10)*5)*_Point;
     }
   else             // ORDER_TYPE_SELL_LIMIT o ORDER_TYPE_BUY_STOP
```

```
      {
       price=last_tick.ask; // partimos del precio Ask
       price=price+(distance+(MathRand()%10)*5)*_Point;
      }
//---
   return(price);
  }
```

Véase también

Tipos de operaciones comerciales, Estructura de solicitud comercial, Estructura de resultado de solicitud comercial

OrderSendAsync

La función OrderSendAsync() sirve para realizar las operaciones de trading asincrónicas, sin esperar la respuesta del servidor comercial a la solicitud enviada. Esta función está diseñada para un trading de alta frecuencia, cuando según las condiciones del algoritmo de trading resulta inadmisible perder el tiempo para esperar la respuesta del servidor.

```
bool  OrderSendAsync(
   MqlTradeRequest&  request,      // estructura de la solicitud
   MqlTradeResult&   result        // estructura de la respuesta
   );
```

Parámetros

request

[in] Puntero a la estructura del tipo MqlTradeRequest que describe la acción comercial del cliente.

result

[in,out] Puntero a la estructura del tipo MqlTradeResult que describe el resultado de la operación comercial cuando la función se finaliza con éxito (si se devuelve true).

Valor devuelto

Devuelve true al enviar con éxito la solicitud comercial al servidor de trading. Si la solicitud no ha sido enviada, devuelve false. En caso de la ejecución con éxito, en la variable *result* el código de la respuesta contiene el valor TRADE_RETCODE_PLACED (código 10008) - "orden colocada". La ejecución con éxito significa sólo el hecho del envío, pero no da ninguna garantía de que la solicitud haya llegado al servidor comercial y haya sido aceptada para la tramitación. Durante el procesamiento de la solicitud recibida el servidor comercial envía al Terminal de Cliente un mensaje de respuesta referente al cambio del estado actual de la posición, órdenes y transacciones que provoca la generación del evento Trade.

En el servidor el resultado de ejecución de la solicitud comercial que ha sido enviada por la función OrderSendAsync() se puede seguir a través del manejador OnTradeTransaction. Hay que tener en cuenta que durante la ejecución de una solicitud comercial el manejador OnTradeTransaction será invocado varias veces.

Por ejemplo, al enviar una orden de compra, ésta se tramita, para la cuenta se crea una orden de compra correspondiente, se realiza la ejecución de la orden, su eliminación de la lista de las abiertas, se agrega al historial de órdenes, luego la operación correspondiente se agrega al historial, y se crea una posición nueva. La función OnTradeTransaction será llamada para cada uno de estos eventos. Para obtener la información más detallada, hay que analizar los parámetros de esta función:

- **trans** - este parámetro obtiene la estructura MqlTradeTransaction que describe la transacción comercial aplicada a la cuenta de trading;
- **request** - este parámetro obtiene la estructura MqlTradeRequest que describe la solicitud comercial que ha provocado la ejecución de la transacción comercial;
- **result** - este parámetro obtiene la estructura MqlTradeResult que describe el resultado de ejecución de la solicitud comercial.

Nota

Esta función en términos del propósito y parámetros es igual a la OrderSend(), pero a diferencia de

aquélla es una versión asincrónica. Es decir, no detiene el trabajo del programa a la espera del resultado de su ejecución. Usted puede comparar la velocidad de operaciones comerciales de estas dos funciones con la ayuda del EA del ejemplo de abajo.

Ejemplo:

```
#property description "EAs para el envío de solicitudes de trading "
                      " a través de la función OrderSendAsync().\r\n"
#property description "Se muestra el procesamiento de eventos de trading mediante"
                      " las funciones-manejadores OnTrade() y OnTradeTransaction()\r\
#property description "En los parámetros del EA se puede establecer el Magic Number"
                      " (identificador único) "
#property description "y el modo de visualización de mensajes en el diario "Asesores
//--- input parameters
input int  MagicNumber=1234567;      // Identificador del EA
input bool DescriptionModeFull=true; // Modo de visualización detallado
//--- variable para el uso en la llamada HistorySelect()
datetime history_start;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- comprobamos el permiso para el trading automático
   if(!TerminalInfoInteger(TERMINAL_TRADE_ALLOWED))
     {
      Alert("El trading automático está prohibido en el terminal, el EA será eliminad
      ExpertRemove();
      return(-1);
     }
//--- no se puede tradear en la cuenta real
   if(AccountInfoInteger(ACCOUNT_TRADE_MODE)==ACCOUNT_TRADE_MODE_REAL)
     {
      Alert("¡El EA tiene prohibido tradear en la cuenta real!");
      ExpertRemove();
      return(-2);
     }
//--- comprobar si se puede tradear en esta cuenta (por ejemplo, el trading es imposi
   if(!AccountInfoInteger(ACCOUNT_TRADE_ALLOWED))
     {
      Alert("Está prohibido tradear en esta cuenta");
      ExpertRemove();
      return(-3);
     }
//--- recordamos la hora de inicio del EA para obtener el historial de trading
   history_start=TimeCurrent();
//---
   CreateBuySellButtons();
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- eliminamos los objetos gráficos
   ObjectDelete(0,"Buy");
   ObjectDelete(0,"Sell");
//---
  }
//+------------------------------------------------------------------+
//| TradeTransaction function                                        |
//+------------------------------------------------------------------+
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
  {
```

```
//--- encabezamiento según el nombre de la función-manejador del evento de trading
   Print("=> ",__FUNCTION__," at ",TimeToString(TimeCurrent(),TIME_SECONDS));
//--- obtenemos el tipo de la transacción como valor de la enumeración
   ENUM_TRADE_TRANSACTION_TYPE type=trans.type;
//--- si la transacción es el resultado del procesamiento de la solicitud
   if(type==TRADE_TRANSACTION_REQUEST)
     {
      //---mostramos el nombre e la transacción
      Print(EnumToString(type));
      //--- luego mostramos la descripción literal de la solicitud procesada
      Print("------------RequestDescription\r\n",
            RequestDescription(request,DescriptionModeFull));
      //--- y mostramos la descripción del resultado de la solicitud
      Print("------------ ResultDescription\r\n",
            TradeResultDescription(result,DescriptionModeFull));
     }
   else // para la transacción del otro tipo mostramos la descripción completa de la
     {
      Print("------------ TransactionDescription\r\n",
            TransactionDescription(trans,DescriptionModeFull));
     }
//---
  }
//+------------------------------------------------------------------+
//| Trade function                                                   |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- miembros estáticos para el almacenaje del estado de la cuenta de trading
   static int prev_positions=0,prev_orders=0,prev_deals=0,prev_history_orders=0;
//--- solicitamos el historial de trading
   bool update=HistorySelect(history_start,TimeCurrent());
   PrintFormat("HistorySelect(%s , %s) = %s",
               TimeToString(history_start),TimeToString(TimeCurrent()),(string)update
//--- encabezamiento según el nombre de la función-manejador del evento de trading
   Print("=> ",__FUNCTION__," at ",TimeToString(TimeCurrent(),TIME_SECONDS));
//--- mostramos el nombre del manejador y el número de órdenes para el momento de pro
   int curr_positions=PositionsTotal();
   int curr_orders=OrdersTotal();
   int curr_deals=HistoryOrdersTotal();
   int curr_history_orders=HistoryDealsTotal();
//--- mostramos el número de órdenes, posiciones, transacciones, asi como los cambios
   PrintFormat("PositionsTotal() = %d (%+d)",
               curr_positions,(curr_positions-prev_positions));
   PrintFormat("OrdersTotal() = %d (%+d)",
               curr_orders,curr_orders-prev_orders);
   PrintFormat("HistoryOrdersTotal() = %d (%+d)",
               curr_deals,curr_deals-prev_deals);
   PrintFormat("HistoryDealsTotal() = %d (%+d)",
               curr_history_orders,curr_history_orders-prev_history_orders);
//--- inserción de string break para leer el Diario con comodidad
   Print("");
//--- guardamos el estado de la cuenta
   prev_positions=curr_positions;
   prev_orders=curr_orders;
   prev_deals=curr_deals;
   prev_history_orders=curr_history_orders;
//---
  }
//+------------------------------------------------------------------+
//| ChartEvent function                                              |
```

```
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam)
  {
//--- procesamiento del evento CHARTEVENT_CLICK ("Clic con el botón del ratón sobre e
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      Print("=> ",__FUNCTION__,": sparam = ",sparam);
      //--- volumen mínimo para la transacción
      double volume_min=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN);
      //--- si está pulsado el botón "Buy", compramos
      if(sparam=="Buy")
        {
         PrintFormat("Buy %s %G lot",_Symbol,volume_min);
         BuyAsync(volume_min);
         //--- despulsamos el botón pulsado
         ObjectSetInteger(0,"Buy",OBJPROP_STATE,false);
        }
      //--- si está pulsado el botón "Sell", vendemos
      if(sparam=="Sell")
        {
         PrintFormat("Sell %s %G lot",_Symbol,volume_min);
         SellAsync(volume_min);
         //--- despulsamos el botón pulsado
         ObjectSetInteger(0,"Sell",OBJPROP_STATE,false);
        }
      ChartRedraw();
     }
//---
  }
//+------------------------------------------------------------------+
//|  Devuelve la descripción literal de la transacción               |
//+------------------------------------------------------------------+
string TransactionDescription(const MqlTradeTransaction &trans,
                              const bool detailed=true)
  {
//--- preparamos la cadena para el retorno desde la función
   string desc=EnumToString(trans.type)+"\r\n";
//--- en el modo detallado agregamos el máximo de información
   if(detailed)
     {
      desc+="Symbol: "+trans.symbol+"\r\n";
      desc+="Deal ticket: "+(string)trans.deal+"\r\n";
      desc+="Deal type: "+EnumToString(trans.deal_type)+"\r\n";
      desc+="Order ticket: "+(string)trans.order+"\r\n";
      desc+="Order type: "+EnumToString(trans.order_type)+"\r\n";
      desc+="Order state: "+EnumToString(trans.order_state)+"\r\n";
      desc+="Order time type: "+EnumToString(trans.time_type)+"\r\n";
      desc+="Order expiration: "+TimeToString(trans.time_expiration)+"\r\n";
      desc+="Price: "+StringFormat("%G",trans.price)+"\r\n";
      desc+="Price trigger: "+StringFormat("%G",trans.price_trigger)+"\r\n";
      desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"\r\n";
      desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"\r\n";
      desc+="Volume: "+StringFormat("%G",trans.volume)+"\r\n";
     }
//--- devolvemos la cadena obtenida
   return desc;
  }
//+------------------------------------------------------------------+
```

```
//|  Devuelve la descripción literal de solicitud comercial                  |
//+------------------------------------------------------------------+
string RequestDescription(const MqlTradeRequest &request,
                          const bool detailed=true)
  {
//--- preparamos la cadena para la devolución desde la función
   string desc=EnumToString(request.action)+"\r\n";
//--- en el modo detallado agregamos el máximo de información
   if(detailed)
     {
      desc+="Symbol: "+request.symbol+"\r\n";
      desc+="Magic Number: "+StringFormat("%d",request.magic)+"\r\n";
      desc+="Order ticket: "+(string)request.order+"\r\n";
      desc+="Order type: "+EnumToString(request.type)+"\r\n";
      desc+="Order filling: "+EnumToString(request.type_filling)+"\r\n";
      desc+="Order time type: "+EnumToString(request.type_time)+"\r\n";
      desc+="Order expiration: "+TimeToString(request.expiration)+"\r\n";
      desc+="Price: "+StringFormat("%G",request.price)+"\r\n";
      desc+="Deviation points: "+StringFormat("%G",request.deviation)+"\r\n";
      desc+="Stop Loss: "+StringFormat("%G",request.sl)+"\r\n";
      desc+="Take Profit: "+StringFormat("%G",request.tp)+"\r\n";
      desc+="Stop Limit: "+StringFormat("%G",request.stoplimit)+"\r\n";
      desc+="Volume: "+StringFormat("%G",request.volume)+"\r\n";
      desc+="Comment: "+request.comment+"\r\n";
     }
//--- devolvemos la cadena obtenida
   return desc;
  }
//+------------------------------------------------------------------+
//|  Devuelve la descripción literal del resultado de tramitación de la solicitud
//+------------------------------------------------------------------+
string TradeResultDescription(const MqlTradeResult &result,
                              const bool detailed=true)
  {
//--- preparamos la cadena para el retorno desde la función
   string desc="Retcode "+(string)result.retcode+"\r\n";
//--- en el modo detallado agregamos el máximo de información
   if(detailed)
     {
      desc+="Request ID: "+StringFormat("%d",result.request_id)+"\r\n";
      desc+="Order ticket: "+(string)result.order+"\r\n";
      desc+="Deal ticket: "+(string)result.deal+"\r\n";
      desc+="Volume: "+StringFormat("%G",result.volume)+"\r\n";
      desc+="Price: "+StringFormat("%G",result.price)+"\r\n";
      desc+="Ask: "+StringFormat("%G",result.ask)+"\r\n";
      desc+="Bid: "+StringFormat("%G",result.bid)+"\r\n";
      desc+="Comment: "+result.comment+"\r\n";
     }
//--- devolvemos la cadena obtenida
   return desc;
  }
//+------------------------------------------------------------------+
//| Crea dos botones para la compra y la venta                       |
//+------------------------------------------------------------------+
void CreateBuySellButtons()
  {
//--- comprobamos la presencia del objeto con el nombre "Buy"
   if(ObjectFind(0,"Buy")>=0)
     {
      //--- si el objeto encontrado no es un botón, lo eliminamos
      if(ObjectGetInteger(0,"Buy",OBJPROP_TYPE)!=OBJ_BUTTON)
```

```
         ObjectDelete(0,"Buy");
      }
   else
      ObjectCreate(0,"Buy",OBJ_BUTTON,0,0,0); // creamos el botón "Buy"
//--- configuramos el botón "Buy"
   ObjectSetInteger(0,"Buy",OBJPROP_CORNER,CORNER_RIGHT_UPPER);
   ObjectSetInteger(0,"Buy",OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,"Buy",OBJPROP_YDISTANCE,50);
   ObjectSetInteger(0,"Buy",OBJPROP_XSIZE,70);
   ObjectSetInteger(0,"Buy",OBJPROP_YSIZE,30);
   ObjectSetString(0,"Buy",OBJPROP_TEXT,"Buy");
   ObjectSetInteger(0,"Buy",OBJPROP_COLOR,clrRed);
//--- comprobamos la presencia del objeto con el nombre "Sell"
   if(ObjectFind(0,"Sell")>=0)
     {
      //--- si el objeto encontrado no es un botón, lo eliminamos
      if(ObjectGetInteger(0,"Sell",OBJPROP_TYPE)!=OBJ_BUTTON)
         ObjectDelete(0,"Sell");
     }
   else
      ObjectCreate(0,"Sell",OBJ_BUTTON,0,0,0); // creamos el botón "Sell"
//--- configuramos el botón "Buy"
   ObjectSetInteger(0,"Sell",OBJPROP_CORNER,CORNER_RIGHT_UPPER);
   ObjectSetInteger(0,"Sell",OBJPROP_XDISTANCE,100);
   ObjectSetInteger(0,"Sell",OBJPROP_YDISTANCE,100);
   ObjectSetInteger(0,"Sell",OBJPROP_XSIZE,70);
   ObjectSetInteger(0,"Sell",OBJPROP_YSIZE,30);
   ObjectSetString(0,"Sell",OBJPROP_TEXT,"Sell");
   ObjectSetInteger(0,"Sell",OBJPROP_COLOR,clrBlue);
//--- forzamos la actualización del gráfico para que los botones se dibujen inmediata
   ChartRedraw();
//---
  }
//+------------------------------------------------------------------+
//| la compra mediante la función asincrónica OrderSendAsync()       |
//+------------------------------------------------------------------+
void BuyAsync(double volume)
  {
//--- preparamos la solicitud
   MqlTradeRequest req={0};
   req.action      =TRADE_ACTION_DEAL;
   req.symbol      =_Symbol;
   req.magic       =MagicNumber;
   req.volume      =0.1;
   req.type        =ORDER_TYPE_BUY;
   req.price       =SymbolInfoDouble(req.symbol,SYMBOL_ASK);
   req.deviation   =10;
   req.comment     ="Buy using OrderSendAsync()";
   MqlTradeResult  res={0};
   if(!OrderSendAsync(req,res))
     {
      Print(__FUNCTION__,": error ",GetLastError(),", retcode = ",res.retcode);
     }
//---
  }
//+------------------------------------------------------------------+
//| la venta mediante la función asincrónica OrderSendAsync()        |
//+------------------------------------------------------------------+
void SellAsync(double volume)
  {
//--- preparamos la solicitud
```

```
   MqlTradeRequest req={0};
   req.action       =TRADE_ACTION_DEAL;
   req.symbol       =_Symbol;
   req.magic        =MagicNumber;
   req.volume       =0.1;
   req.type         =ORDER_TYPE_SELL;
   req.price        =SymbolInfoDouble(req.symbol,SYMBOL_BID);
   req.deviation    =10;
   req.comment      ="Sell using OrderSendAsync()";
   MqlTradeResult  res={0};
   if(!OrderSendAsync(req,res))
     {
      Print(__FUNCTION__,": error ",GetLastError(),", retcode = ",res.retcode);
     }
//---
  }
//+------------------------------------------------------------------+
```

Ejemplo de visualización de mensajes en el diario "Asesores Expertos":

```
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnChartEvent: sparam = Sell
12:52:52   ExpertAdvisor (EURUSD,H1)   Sell EURUSD 0.01 lot
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_REQUEST
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------RequestDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_ACTION_DEAL
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Magic Number: 1234567
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order filling: ORDER_FILLING_FOK
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Deviation points: 10
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Limit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)   Comment: Sell using OrderSendAsync()
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ ResultDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   Retcode 10009
12:52:52   ExpertAdvisor (EURUSD,H1)   Request ID: 2
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 15048668
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Ask: 1.29319
12:52:52   ExpertAdvisor (EURUSD,H1)   Bid: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Comment:
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+1)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+2)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+2)
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_ORDER_ADD
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_STARTED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_ORDER_DELETE
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 0
```

```
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_STARTED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_HISTORY_ADD
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_FILLED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTradeTransaction at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   ------------ TransactionDescription
12:52:52   ExpertAdvisor (EURUSD,H1)   TRADE_TRANSACTION_DEAL_ADD
12:52:52   ExpertAdvisor (EURUSD,H1)   Symbol: EURUSD
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal ticket: 15048668
12:52:52   ExpertAdvisor (EURUSD,H1)   Deal type: DEAL_TYPE_SELL
12:52:52   ExpertAdvisor (EURUSD,H1)   Order ticket: 16361998
12:52:52   ExpertAdvisor (EURUSD,H1)   Order type: ORDER_TYPE_BUY
12:52:52   ExpertAdvisor (EURUSD,H1)   Order state: ORDER_STATE_STARTED
12:52:52   ExpertAdvisor (EURUSD,H1)   Order time type: ORDER_TIME_GTC
12:52:52   ExpertAdvisor (EURUSD,H1)   Order expiration: 1970.01.01 00:00
12:52:52   ExpertAdvisor (EURUSD,H1)   Price: 1.29313
12:52:52   ExpertAdvisor (EURUSD,H1)   Price trigger: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Stop Loss: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Take Profit: 0
12:52:52   ExpertAdvisor (EURUSD,H1)   Volume: 0.1
12:52:52   ExpertAdvisor (EURUSD,H1)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistorySelect( 09:34 , 09:52) = true
```

```
12:52:52   ExpertAdvisor (EURUSD,H1)   => OnTrade at 09:52:53
12:52:52   ExpertAdvisor (EURUSD,H1)   PositionsTotal() = 1 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   OrdersTotal() = 0 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryOrdersTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)   HistoryDealsTotal() = 2 (+0)
12:52:52   ExpertAdvisor (EURUSD,H1)
```

PositionsTotal

Devuelve el número de posiciones abiertas.

```
int  PositionsTotal();
```

Valor devuelto

Valor del tipo int.

Nota

En cualquier período de tiempo para cada símbolo se puede abrir sólo una posición, la cual puede ser el resultado de una o más transacciones. No se puede confundir las posiciones con las órdenes pendientes que también se muestran en la pestaña "Operaciones" del panel de herramientas del terminal de cliente.

El número total de posiciones de una cuenta comercial no puede superar el número total de instrumentos financieros.

Véase también

PositionGetSymbol(), PositionSelect(), Propiedades de posiciones

PositionGetSymbol

Devuelve el símbolo de una posición correspondiente abierta y automáticamente elige una posición para gestionarla después usando las funciones PositionGetDouble,PositionGetInteger, PositionGetString.

```
string  PositionGetSymbol(
   int  index      // número en la lista de posiciones
   );
```

Parámetros

index

[in] Número de posición en la lista de posiciones abiertas.

Valor devuelto

Valor del tipo string.

Nota

En cualquier período de tiempo para cada símbolo se puede abrir sólo una posición, la cual puede ser el resultado de una o más transacciones. No se puede confundir las posiciones con las órdenes pendientes que también se muestran en la pestaña "Operaciones" del panel de herramientas del terminal de cliente.

El número total de posiciones de una cuenta comercial no puede superar el número total de instrumentos financieros.

Véase también

PositionsTotal(), PositionSelect(), Propiedades de posiciones

PositionSelect

Elige una posición abierta para el futuro trabajo con ella. Devuelve true en caso de que la ejecución de la función se finalice con éxito, de lo contrario devuelve false. Para obtener la información sobre el error hay que llamar a la función GetLastError().

```
bool  PositionSelect(
   string  symbol,     // nombre del instrumento
   );
```

Parámetros

symbol

[in] denominación del instrumento financiero.

Valor devuelto

Valor del tipo bool.

Nota

En cualquier período de tiempo para cada símbolo se puede abrir sólo una posición, la cual puede ser el resultado de una o más transacciones. No se puede confundir las posiciones con las órdenes pendientes que también se muestran en la pestaña "Operaciones" del panel de herramientas del terminal de cliente.

La función PositionSelect() copia los datos sobre la posición en el entorno del programa, y las posteriores llamadas PositionGetDouble(), PositionGetInteger() y PositionGetString() devuelven los datos copiados anteriormente. Eso significa que la propia posición a lo mejor ya no existe (o tal vez se haya cambiado su volumen, dirección, etc.), pero todavía se puede seguir recibiendo los datos sobre esta posición. Para garantizar la recepción de datos recientes sobre una posición, se recomienda llamar a la función PositionSelect() justamente antes de solicitarlos.

Véase también

PositionGetSymbol(), PositionsTotal(), Propiedades de posiciones

PositionGetDouble

La función devuelve la propiedad solicitada de una posición abierta que previamente ha sido elegida a través de la función PositionGetSymbol o PositionSelect. La propiedad de la posición tiene que ser del tipo double. Existen 2 variantes de la función.

1. Inmediatamente devuelve el valor de la propiedad.

```
double  PositionGetDouble(
   ENUM_POSITION_PROPERTY_DOUBLE  property_id      // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  PositionGetDouble(
   ENUM_POSITION_PROPERTY_DOUBLE  property_id,     // identificador de la propiedad
   double&                        double_var       // aquí recibimos el valor de la p
   );
```

Parámetros

property_id

[in] Identificador de la propiedad de la posición. Su valor puede ser uno de los valores de la enumeración ENUM_POSITION_PROPERTY_DOUBLE.

double_var

[out] Variable del tipo double que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo double.

Nota

En cualquier período de tiempo para cada símbolo se puede abrir sólo una posición, la cual puede ser el resultado de una o más transacciones. No se puede confundir las posiciones con las órdenes pendientes que también se muestran en la pestaña "Operaciones" del panel de herramientas del terminal de cliente.

Para garantizar la recepción de datos recientes sobre una posición, se recomienda llamar a la función PositionSelect() justamente antes de solicitarlos.

Véase también

PositionGetSymbol(), PositionSelect(), Propiedades de posiciones

PositionGetInteger

La función devuelve la propiedad solicitada de una posición abierta que previamente ha sido elegida a través de la función PositionGetSymbol o PositionSelect. La propiedad de la posición tiene que ser del tipo datetime, int. Existen 2 variantes de la función.

1. Inmediatamente devuelve el valor de la propiedad.

```
long  PositionGetInteger(
   ENUM_POSITION_PROPERTY_INTEGER  property_id      // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  PositionGetInteger(
   ENUM_POSITION_PROPERTY_INTEGER  property_id,     // identificador de la propiedad
   long&                           long_var         // aquí recibimos el valor de la
   );
```

Parámetros

property_id

[in] Identificador de la propiedad de la posición. Su valor puede ser uno de los valores de la enumeración ENUM_POSITION_PROPERTY_INTEGER.

long_var

[out] Variable del tipo long que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo long. En caso de ejecución fallida devuelve 0.

Nota

En cualquier período de tiempo para cada símbolo se puede abrir sólo una posición, la cual puede ser el resultado de una o más transacciones. No se puede confundir las posiciones con las órdenes pendientes que también se muestran en la pestaña "Operaciones" del panel de herramientas del terminal de cliente.

Para garantizar la recepción de datos recientes sobre una posición, se recomienda llamar a la función PositionSelect() justamente antes de solicitarlos.

Ejemplo:

```
//+------------------------------------------------------------------+
//| Trade function                                                   |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- comprobamos si la posición está presente y mostramos el tiempo de su cambio
   if(PositionSelect(_Symbol))
     {
//--- obtenemos el identificador de la posición para poder seguir trabajando con ella
      ulong position_ID=PositionGetInteger(POSITION_IDENTIFIER);
      Print(_Symbol," postion #",position_ID);
//--- obtenemos el tiempo de formación de la posición en milisegundos pasados desde e
      long create_time_msc=PositionGetInteger(POSITION_TIME_MSC);
      PrintFormat("Position #%d  POSITION_TIME_MSC = %i64 milliseconds => %s",positio
                  create_time_msc,TimeToString(create_time_msc/1000));
//--- obtenemos el tiempo del último cambio de la posición en segundos desde el 01.01
      long update_time_sec=PositionGetInteger(POSITION_TIME_UPDATE);
      PrintFormat("Position #%d  POSITION_TIME_UPDATE = %i64 seconds => %s",
                  position_ID,update_time_sec,TimeToString(update_time_sec));
//--- obtenemos el tiempo del último cambio de la posición en milisegundos desde el 0
      long update_time_msc=PositionGetInteger(POSITION_TIME_UPDATE_MSC);
      PrintFormat("Position #%d  POSITION_TIME_UPDATE_MSC = %i64 milliseconds => %s",
                  position_ID,update_time_msc,TimeToString(update_time_msc/1000));
     }
//---
  }
```

Véase también

PositionGetSymbol(), PositionSelect(), Propiedades de posiciones

PositionGetString

La función devuelve la propiedad solicitada de una posición abierta que previamente ha sido elegida a través de la función PositionGetSymbol o PositionSelect. La propiedad de la posición tiene que ser del tipo string. Existen 2 variantes de la función.

1. Inmediatamente devuelve el valor de la propiedad.

```
string  PositionGetString(
   ENUM_POSITION_PROPERTY_STRING  property_id      // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  PositionGetString(
   ENUM_POSITION_PROPERTY_STRING  property_id,     // identificador de la propiedad
   string&                        string_var       // aquí recibimos el valor de la p
   );
```

Parámetros

property_id

[in] Identificador de la propiedad de la posición. Su valor puede ser uno de los valores de la enumeración ENUM_POSITION_PROPERTY_STRING.

string_var

[out] Variable del tipo string que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo string.

Nota

En cualquier período de tiempo para cada símbolo se puede abrir sólo una posición, la cual puede ser el resultado de una o más transacciones. No se puede confundir las posiciones con las órdenes pendientes que también se muestran en la pestaña "Operaciones" del panel de herramientas del terminal de cliente.

Para garantizar la recepción de datos recientes sobre una posición, se recomienda llamar a la función PositionSelect() justamente antes de solicitarlos.

Véase también

PositionGetSymbol(), PositionSelect(), Propiedades de posiciones

OrdersTotal

Devuelve el número de órdenes.

```
int  OrdersTotal();
```

Valor devuelto

Valor del tipo int.

Nota

No se puede confundir las órdenes pendientes con las posiciones que también se muestran en la pestaña "Operaciones" del panel "Caja de Herramientas" del terminal de cliente. Una orden es la indicación de realizar una operación comercial, mientras que una posición es el resultado de una o varias transacciones.

En cualquier momento para cada símbolo se puede abrir sólo una posición, mientras que puede haber varias órdenes pendientes para el mismo símbolo.

Véase también

OrderSelect(), OrderGetTicket(), Propiedades de órdenes

OrderGetTicket

Devuelve el ticket de una orden correspondiente y automáticamente elige la orden para trabajar con ella después usando las funciones.

```
ulong  OrderGetTicket(
   int  index      // número en la lista de órdenes
   );
```

Parámetros

index

[in] Número de una orden en la lista de órdenes

Valor devuelto

Valor del tipo ulong. En caso de ejecución fallida devuelve 0.

Nota

No se puede confundir las órdenes pendientes con las posiciones que también se muestran en la pestaña "Operaciones" del panel "Caja de Herramientas" del terminal de cliente. Una orden es la indicación de realizar una operación comercial, mientras que una posición es el resultado de una o varias transacciones.

En cualquier momento para cada símbolo se puede abrir sólo una posición, mientras que puede haber varias órdenes pendientes para el mismo símbolo.

La función OrderGetTicket() copia los datos sobre la orden en el entorno del programa, y las posteriores llamadas OrderGetDouble(), OrderGetInteger(), OrderGetString() devuelven los datos copiados anteriormente. Eso significa que la propia orden a lo mejor ya no existe (o se ha cambiado su precio de apertura, niveles Stop Loss / Take Profit o el plazo de vencimiento), pero todavía se puede seguir recibiendo los datos sobre esta orden. Para garantizar la recepción de datos recientes sobre una orden, se recomienda llamar a la función OrderGetTicket() justamente antes de solicitarlos.

Ejemplo:

```
void OnStart()
  {
//--- variables para recibir valores desde las propiedades de la orden
   ulong    ticket;
   double   open_price;
   double   initial_volume;
   datetime time_setup;
   string   symbol;
   string   type;
   long     order_magic;
   long     positionID;
//--- número de actuales órdenes pendientes
   uint     total=OrdersTotal();
//--- repasamos todas las órdenes en el ciclo
   for(uint i=0;i<total;i++)
```

```
     {
      //--- recibimos el ticket de la orden según su posición en la lista
      if((ticket=OrderGetTicket(i))>0)
        {
         //--- recibimos las propiedades de la orden
         open_price=   =OrderGetDouble(ORDER_PRICE_OPEN);
         time_setup    =OrderGetInteger(ORDER_TIME_SETUP);
         symbol        =OrderGetString(ORDER_SYMBOL);
         order_magic   =OrderGetInteger(ORDER_MAGIC);
         positionID    =OrderGetInteger(ORDER_POSITION_ID);
         initial_volume=OrderGetDouble(ORDER_VOLUME_INITIAL);
         type          =EnumToString(ORDER_TYPE);
         //--- preparamos y mostramos información sobre la orden
         printf("#ticket %d %s %G %s at %G was set up at %s",
                ticket,                 // ticket de la orden
                type,                   // tipo
                initial_volume,         // volumen colocado
                symbol,                 // símbolo
                open_price,             // precio de apertura especificado
                TimeToString(time_setup)// hora de colocación de la orden
                );
        }
     }
//---
  }
```

Véase también

OrdersTotal(), OrderSelect(), OrderGetInteger()

OrderSelect

Elige una orden para el futuro trabajo con ella. Devuelve true en caso de que la ejecución de la función se finalice con éxito, de lo contrario devuelve false. Para obtener la información sobre el error hay que llamar a la función GetLastError().

```
bool  OrderSelect(
   ulong    ticket,      // ticket de la orden
   );
```

Parámetros

ticket

[in] Ticket de la orden.

Valor devuelto

Valor del tipo bool.

Nota

No se puede confundir las órdenes pendientes con las posiciones que también se muestran en la pestaña "Operaciones" del panel "Caja de Herramientas" del terminal de cliente. En cualquier momento para cada símbolo se puede abrir sólo una posición, mientras que puede haber varias órdenes pendientes para el mismo símbolo.

La función OrderSelect() copia los datos sobre la orden en el entorno del programa, y las posteriores llamadas OrderGetDouble(), OrderGetInteger(), OrderGetString() devuelven los datos copiados anteriormente. Eso significa que la propia orden a lo mejor ya no existe (o se ha cambiado su precio de apertura, niveles Stop Loss / Take Profit o el plazo de vencimiento), pero todavía se puede seguir recibiendo los datos sobre esta orden. Para garantizar la recepción de datos recientes sobre una orden, se recomienda llamar a la función OrderSelect() justamente antes de solicitarlos.

Véase también

OrderGetInteger(), OrderGetDouble(), OrderGetString(), OrderGetTicket(), Propiedades de órdenes

OrderGetDouble

La función devuelve la propiedad solicitada de una orden que previamente ha sido elegida a través de la función OrderGetTicket o OrderSelect. La propiedad de la orden tiene que ser del tipo double. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
double  OrderGetDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  property_id        // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  OrderGetDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  property_id,      // identificador de la propiedad
   double&                     double_var        // aquí recibimos el valor de la pro
   );
```

Parámetros

property_id

[in] Identificador de la propiedad de la orden. Su valor puede ser uno de los valores de la enumeración ENUM_ORDER_PROPERTY_DOUBLE.

double_var

[out] Variable del tipo double que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo double.

Nota

No se puede confundir las órdenes pendientes con las posiciones que también se muestran en la pestaña "Operaciones" del panel "Caja de Herramientas" del terminal de cliente. En cualquier momento para cada símbolo se puede abrir sólo una posición, mientras que puede haber varias órdenes pendientes para el mismo símbolo.

Para garantizar la recepción de datos recientes sobre una orden, se recomienda llamar a la función OrderSelect() justamente antes de solicitarlos.

Véase también

OrdersTotal(), OrderGetTicket(), Propiedades de órdenes

OrderGetInteger

La función devuelve la propiedad solicitada de una orden que previamente ha sido elegida a través de la función OrderGetTicket o OrderSelect. La propiedad de la orden tiene que ser del tipo datetime, int. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
long  OrderGetInteger(
   ENUM_ORDER_PROPERTY_INTEGER  property_id        // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  OrderGetInteger(
   ENUM_ORDER_PROPERTY_INTEGER  property_id,      // identificador de la propiedad
   long&                        long_var          // aquí recibimos el valor de la pr
   );
```

Parámetros

property_id

[in] Identificador de la propiedad de la orden. Su valor puede ser uno de los valores de la enumeración ENUM_ORDER_PROPERTY_INTEGER.

long_var

[out] Variable del tipo long que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo long.

Nota

No se puede confundir las órdenes pendientes con las posiciones que también se muestran en la pestaña "Operaciones" del panel "Caja de Herramientas" del terminal de cliente. En cualquier momento para cada símbolo se puede abrir sólo una posición, mientras que puede haber varias órdenes pendientes para el mismo símbolo.

Para garantizar la recepción de datos recientes sobre una orden, se recomienda llamar a la función OrderSelect() justamente antes de solicitarlos.

Véase también

OrdersTotal(), OrderGetTicket(), Propiedades de órdenes

OrderGetString

La función devuelve la propiedad solicitada de una orden que previamente ha sido elegida a través de la función OrderGetTicket o OrderSelect. La propiedad de la orden tiene que ser del tipo string. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
string  OrderGetString(
   ENUM_ORDER_PROPERTY_STRING  property_id        // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  OrderGetString(
   ENUM_ORDER_PROPERTY_STRING  property_id,      // identificador de la propiedad
   string&                     string_var        // aquí recibimos el valor de la pro
   );
```

Parámetros

property_id

[in] Identificador de la propiedad de la orden. Su valor puede ser uno de los valores de la enumeración ENUM_ORDER_PROPERTY_STRING.

string_var

[out] Variable del tipo strng que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo string.

Nota

No se puede confundir las órdenes pendientes con las posiciones que también se muestran en la pestaña "Operaciones" del panel "Caja de Herramientas" del terminal de cliente. En cualquier momento para cada símbolo se puede abrir sólo una posición, mientras que puede haber varias órdenes pendientes para el mismo símbolo.

Para garantizar la recepción de datos recientes sobre una orden, se recomienda llamar a la función OrderSelect() justamente antes de solicitarlos.

Véase también

OrdersTotal(), OrderGetTicket(), Propiedades de órdenes

HistorySelect

Solicita el historial de transacciones y órdenes del período especificado de la hora del servidor.

```
bool  HistorySelect(
   datetime  from_date,     // desde la fecha
   datetime  to_date        // hasta la fecha
   );
```

Parámetros

from_date

[in] Fecha inicial de solicitud.

to_date

[in] Fecha final de solicitud.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

La función HistorySelect() crea en el programa mql5 la lista de órdenes y la de transacciones para recurrir posteriormente a estos elementos a través de las funciones correspondientes. El tamaño de la lista de transacciones se puede saber usando la función HistoryDealsTotal(), el tamaño de la lista de órdenes en el historial se puede averiguar con HistoryOrdersTotal(). La selección de los elementos de la lista de órdenes es mejor realizar usando la función HistoryOrderGetTicket(), para los elementos de la lista de transacciones mejor conviene la función HistoryDealGetTicket().

Después de usar la función HistoryOrderSelect() la lista de órdenes en el historial, que están disponibles para el programa mql5, se pone a cero y se llena de nuevo con la orden encontrada, si la búsqueda de orden por el ticket ha sido completada con éxito. Lo mismo se refiere a la lista de transacciones disponibles en el programa mql5, es decir, se pone a cero con la función HistoryDealSelect() y vuelve a llenarse en caso de recibir con éxito una transacción por el número del ticket.

Ejemplo:

```
void OnStart()
  {
   color BuyColor =clrBlue;
   color SellColor=clrRed;
//--- request trade history
   HistorySelect(0,TimeCurrent());
//--- create objects
   string   name;
   uint     total=HistoryDealsTotal();
   ulong    ticket=0;
   double   price;
   double   profit;
   datetime time;
   string   symbol;
```

```
   long     type;
   long     entry;
//--- for all deals
   for(uint i=0;i<total;i++)
     {
      //--- try to get deals ticket
      if((ticket=HistoryDealGetTicket(i))>0)
        {
         //--- get deals properties
         price =HistoryDealGetDouble(ticket,DEAL_PRICE);
         time  =(datetime)HistoryDealGetInteger(ticket,DEAL_TIME);
         symbol=HistoryDealGetString(ticket,DEAL_SYMBOL);
         type  =HistoryDealGetInteger(ticket,DEAL_TYPE);
         entry =HistoryDealGetInteger(ticket,DEAL_ENTRY);
         profit=HistoryDealGetDouble(ticket,DEAL_PROFIT);
         //--- only for current symbol
         if(price && time && symbol==Symbol())
           {
            //--- create price object
            name="TradeHistory_Deal_"+string(ticket);
            if(entry) ObjectCreate(0,name,OBJ_ARROW_RIGHT_PRICE,0,time,price,0,0);
            else      ObjectCreate(0,name,OBJ_ARROW_LEFT_PRICE,0,time,price,0,0);
            //--- set object properties
            ObjectSetInteger(0,name,OBJPROP_SELECTABLE,0);
            ObjectSetInteger(0,name,OBJPROP_BACK,0);
            ObjectSetInteger(0,name,OBJPROP_COLOR,type?BuyColor:SellColor);
            if(profit!=0) ObjectSetString(0,name,OBJPROP_TEXT,"Profit: "+string(profi
           }
        }
     }
//--- apply on chart
   ChartRedraw();
  }
```

Véase también

HistoryOrderSelect(), HistoryDealSelect()

HistorySelectByPosition

Busca el historial de transacciones y órdenes que tienen el identificador de posición especificado.

```
bool  HistorySelectByPosition(
   long   position_id     // identificador de posición - POSITION_IDENTIFIER
   );
```

Parámetros

position_id

[in] Identificador de posición que se asigna a cada orden ejecutada y en cada transacción.

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

No se debe confundir las órdenes del historial de trading con las órdenes pendientes actuales que se muestran en la pestaña "Trading" del panel "Caja de Herramientas" del terminal de cliente. La lista de las órdenes que han sido canceladas o las que han llevado a la ejecución de una operación comercial se puede ver en la pestaña "Historial" del panel "Caja de Herramientas" del terminal de cliente.

La función HistorySelectByPosition() crea la lista de órdenes y la lista de transacciones con el identificador de posición especificado para recurrir posteriormente a estos elementos a través de las funciones correspondientes. El tamaño de la lista de transacciones se puede saber usando la función HistoryDealsTotal(), el tamaño de la lista de órdenes en el historial se puede obtener con HistoryOrdersTotal(). La selección de los elementos de la lista de órdenes es mejor realizar usando la función HistoryOrderGetTicket(), para los elementos de la lista de transacciones mejor conviene la función HistoryDealGetTicket().

Después de usar la función HistoryOrderSelect() la lista de órdenes en el historial, disponibles para el programa mql5, se pone a cero y se llena de nuevo con la orden encontrada, si la búsqueda de orden por el ticket ha sido completada con éxito. Lo mismo se refiere a la lista de transacciones disponibles en el programa mql5, es decir, se pone a cero con la función HistoryDealSelect() y vuelve a llenarse en caso de recibir con éxito una transacción por el número del ticket.

See also

HistorySelect(), HistoryOrderGetTicket(), Order Properties

HistoryOrderSelect

Elige en el historial una orden para recurrir posteriormente a ésta a través de las funciones correspondientes. Devuelve true en caso de que la ejecución de la función se finalice con éxito, de lo contrario devuelve false. Para obtener la información sobre el error hay que llamar a la función GetLastError().

```
bool  HistoryOrderSelect(
   ulong  ticket,      // ticket de orden
   );
```

Parámetros

ticket

[in] Ticket de la orden.

Valor devuelto

Devuelve true en caso del éxito, de lo contrario devuelve false.

Nota

No se debe confundir las órdenes del historial de trading con las órdenes pendientes actuales que se muestran en la pestaña "Trading" del panel "Caja de Herramientas" del terminal de cliente. La lista de las órdenes que han sido canceladas o las que han llevado a la ejecución de una operación comercial se puede ver en la pestaña "Historial" del panel "Caja de Herramientas" del terminal de cliente.

La función HistoryOrderSelect() limpia en un programa mql5 la lista de órdenes del historial, las que están disponible, y copia en esta lista sólo una orden, si la ejecución de la función HistoryOrderSelect() se ha completado con éxito. Si hace falta repasar todas las transacciones seleccionadas por la función HistorySelect() es mejor utilizar la función HistoryOrderGetTicket().

Véase también

HistorySelect(), HistoryOrderGetTicket(), Propiedades de órdenes

HistoryOrdersTotal

Devuelve el número de órdenes en el historial.

```
int  HistoryOrdersTotal();
```

Valor devuelto

Valor del tipo int.

Nota

No se debe confundir las órdenes del historial de trading con las órdenes pendientes actuales que se muestran en la pestaña "Trading" del panel "Caja de Herramientas" del terminal de cliente. La lista de las órdenes que han sido canceladas o las que han llevado a la ejecución de una operación comercial se puede ver en la pestaña "Historial" del panel "Caja de Herramientas" del terminal de cliente.

Véase también

HistorySelect(), HistoryOrderSelect(), HistoryOrderGetTicket(), Propiedades de órdenes

HistoryOrderGetTicket

Devuelve el ticket de una orden correspondiente en el historial.

```
ulong  HistoryOrderGetTicket(
   int  index      // número en la lista de órdenes
   );
```

Parámetros

index

[in] Número de la orden en la lista de órdenes.

Valor devuelto

Valor del tipo ulong. En caso de ejecución fallida devuelve 0.

Nota

No se debe confundir las órdenes del historial de trading con las órdenes pendientes actuales que se muestran en la pestaña "Trading" del panel "Caja de Herramientas" del terminal de cliente. La lista de las órdenes que han sido canceladas o las que han llevado a la ejecución de una operación comercial se puede ver en la pestaña "Historial" del panel "Caja de Herramientas" del terminal de cliente.

Ejemplo:

```
void OnStart()
  {
   datetime from=0;
   datetime to=TimeCurrent();
//--- solicitar todo el historial
   HistorySelect(from,to);
//--- variables para recibir valores desde las propiedades de la orden
   ulong    ticket;
   double   open_price;
   double   initial_volume;
   datetime time_setup;
   datetime time_done;
   string   symbol;
   string   type;
   long     order_magic;
   long     positionID;
//--- número de actuales órdenes pendientes
   uint     total=HistoryOrdersTotal();
//--- repasamos todas las órdenes en el ciclo
   for(uint i=0;i<total;i++)
     {
      //--- recibimos el ticket de la orden según su posición en la lista
      if((ticket=HistoryOrderGetTicket(i))>0)
        {
         //--- recibimos las propiedades de la orden
```

```
         open_price=       HistoryOrderGetDouble(ticket,ORDER_PRICE_OPEN);
         time_setup=       (datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_SETUP);
         time_done=        (datetime)HistoryOrderGetInteger(ticket,ORDER_TIME_DONE);
         symbol=           HistoryOrderGetString(ticket,ORDER_SYMBOL);
         order_magic=      HistoryOrderGetInteger(ticket,ORDER_MAGIC);
         positionID =      HistoryOrderGetInteger(ticket,ORDER_POSITION_ID);
         initial_volume=   HistoryOrderGetDouble(ticket,ORDER_VOLUME_INITIAL);
         type=GetOrderType(HistoryOrderGetInteger(ticket,ORDER_TYPE));
         //--- preparamos y mostramos información sobre la orden
         printf("#ticket %d %s %G %s at %G was set up at %s => done at %s, pos ID=%d"
                ticket,                   // ticket de la orden
                type,                     // tipo
                initial_volume,           // volumen colocado
                symbol,                   // símbolo
                open_price,               // precio de apertura especificado
                TimeToString(time_setup),// hora de colocación de la orden
                TimeToString(time_done), // hora de ejecución y eliminación
                positionID                // ID de la posición en la que ha sido inclu
                );
        }
     }
//---
  }
//+------------------------------------------------------------------+
//|  devuelve el nombre literal del tipo de la orden                 |
//+------------------------------------------------------------------+
string GetOrderType(long type)
  {
   string str_type="unknown operation";
   switch(type)
     {
      case (ORDER_TYPE_BUY):            return("buy");
      case (ORDER_TYPE_SELL):           return("sell");
      case (ORDER_TYPE_BUY_LIMIT):      return("buy limit");
      case (ORDER_TYPE_SELL_LIMIT):     return("sell limit");
      case (ORDER_TYPE_BUY_STOP):       return("buy stop");
      case (ORDER_TYPE_SELL_STOP):      return("sell stop");
      case (ORDER_TYPE_BUY_STOP_LIMIT): return("buy stop limit");
      case (ORDER_TYPE_SELL_STOP_LIMIT):return("sell stop limit");
     }
   return(str_type);
  }
```

Véase también

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), Propiedades de órdenes

HistoryOrderGetDouble

La función devuelve la propiedad de la orden que ha sido solicitada. La propiedad de la orden tiene que ser del tipo double. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
double  HistoryOrderGetDouble(
   ulong                      ticket_number,     // ticket
   ENUM_ORDER_PROPERTY_DOUBLE  property_id       // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  HistoryOrderGetDouble(
   ulong                      ticket_number,     // ticket
   ENUM_ORDER_PROPERTY_DOUBLE  property_id,      // identificador de la propiedad
   double&                    double_var         // aquí recibimos el valor de la pr
   );
```

Parámetros

ticket_number

[in] Ticket de orden.

property_id

[in] Identificador de la propiedad de orden. Su valor puede ser uno de los valores de la enumeración ENUM_ORDER_PROPERTY_DOUBLE.

double_var

[out] Variable del tipo double que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo double.

Véase también

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), Propiedades de órdenes

HistoryOrderGetInteger

La función devuelve la propiedad de la orden que ha sido solicitada. La propiedad de la orden tiene que ser del tipo datetime, int. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
long  HistoryOrderGetInteger(
   ulong                        ticket_number,     // ticket
   ENUM_ORDER_PROPERTY_INTEGER  property_id        // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  HistoryOrderGetInteger(
   ulong                        ticket_number,     // ticket
   ENUM_ORDER_PROPERTY_INTEGER  property_id,       // identificador de la propiedad
   long&                        long_var           // aquí recibimos el valor de la p
   );
```

Parámetros

ticket_number

[in] Ticket de orden.

property_id

[in] Identificador de la propiedad de orden. Su valor puede ser uno de los valores de la enumeración ENUM_ORDER_PROPERTY_INTEGER.

long_var

[out] Variable del tipo long que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo long.

Nota

No se debe confundir las órdenes del historial de trading con las órdenes pendientes actuales que se muestran en la pestaña "Trading" del panel "Caja de Herramientas" del terminal de cliente. La lista de las órdenes que han sido canceladas o las que han llevado a la ejecución de una operación comercial se puede ver en la pestaña "Historial" del panel "Caja de Herramientas" del terminal de cliente.

Ejemplo:

```
//+------------------------------------------------------------------+
//| Trade function                                                   |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- obtenemos el ticket de la última orden desde el historial de trading semanal
   ulong last_order=GetLastOrderTicket();
   if(HistoryOrderSelect(last_order))
     {
      //--- tiempo de colocación de la orden en milisegundos desde el 01.01.1970
      long time_setup_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_SETUP_MSC);
      PrintFormat("Order #%d ORDER_TIME_SETUP_MSC=%i64 => %s",
                  last_order,time_setup_msc,TimeToString(time_setup_msc/1000));
      //--- tiempo de ejecución/eliminación de la orden en milisegundos desde el 01.0
      long  time_done_msc=HistoryOrderGetInteger(last_order,ORDER_TIME_DONE_MSC);
      PrintFormat("Order #%d ORDER_TIME_DONE_MSC=%i64 => %s",
                  last_order,time_done_msc,TimeToString(time_done_msc/1000));
     }
   else // avisamos sobre el fracaso
      PrintFormat("HistoryOrderSelect() failed for #%d. Eror code=%d",
                  last_order,GetLastError());

//---
  }
//+------------------------------------------------------------------+
//|  devuelve el ticket de la última orden en el historial o -1      |
//+------------------------------------------------------------------+
ulong GetLastOrderTicket()
  {
//--- solicitamos el historial de lo últimos 7 días
   if(!GetTradeHistory(7))
     {
      //--- avisamos sobre la llamad fallida y devolveremos -1
      Print(__FUNCTION__," HistorySelect() ha devuelto false");
      return -1;
     }
//---
   ulong first_order,last_order,orders=HistoryOrdersTotal();
//--- si hay órdenes, empezamos a trabajar con ellas
   if(orders>0)
     {
      Print("Orders = ",orders);
      first_order=HistoryOrderGetTicket(0);
      PrintFormat("first_order = %d",first_order);
      if(orders>1)
        {
         last_order=HistoryOrderGetTicket((int)orders-1);
         PrintFormat("last_order = %d",last_order);
         return last_order;
        }
      return first_order;
     }
//--- no hemos encontrado ninguna orden, devolveremos -1
   return -1;
  }
//+------------------------------------------------------------------+
//| solicita el historial de los últimos días y devuelve false en caso de fallo  |
//+------------------------------------------------------------------+
bool GetTradeHistory(int days)
  {
//--- fijamos un período de tiempo semanal para solicitar el historial de trading
```

```
   datetime to=TimeCurrent();
   datetime from=to-days*PeriodSeconds(PERIOD_D1);
   ResetLastError();
//--- hacemos la solicitud y comprobamos el resultado
   if(!HistorySelect(from,to))
     {
      Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());
      return false;
     }
//--- historial recibido con éxito
   return true;
  }
```

Véase también

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), Propiedades de órdenes

HistoryOrderGetString

La función devuelve la propiedad de la orden que ha sido solicitada. La propiedad de la orden tiene que ser del tipo string. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
string  HistoryOrderGetString(
   ulong                      ticket_number,     // ticket
   ENUM_ORDER_PROPERTY_STRING  property_id       // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  HistoryOrderGetString(
   ulong                      ticket_number,     // ticket
   ENUM_ORDER_PROPERTY_STRING  property_id,      // identificador de la propiedad
   string&                    string_var         // aquí recibimos el valor de la pr
   );
```

Parámetros

ticket_number

[in] Ticket de orden.

property_id

[in] Identificador de la propiedad de orden. Su valor puede ser uno de los valores de la enumeración ENUM_ORDER_PROPERTY_STRING.

string_var

[out] Variable del tipo string que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo string.

Nota

No se debe confundir las órdenes del historial de trading con las órdenes pendientes actuales que se muestran en la pestaña "Trading" del panel "Caja de Herramientas" del terminal de cliente. La lista de las órdenes que han sido canceladas o las que han llevado a la ejecución de una operación comercial se puede ver en la pestaña "Historial" del panel "Caja de Herramientas" del terminal de cliente.

Véase también

HistorySelect(), HistoryOrdersTotal(), HistoryOrderSelect(), Propiedades de órdenes

HistoryDealSelect

Elige en el historial una transacción para recurrir posteriormente a ésta a través de las funciones correspondientes. Devuelve true en caso de que la ejecución de la función se finalice con éxito, de lo contrario devuelve false. Para obtener la información sobre el error hay que llamar a la función GetLastError().

```
bool  HistoryDealSelect(
   ulong  ticket,     // ticket de la transacción
   );
```

Parámetros

ticket

[in] Ticket de la transacción.

Valor devuelto

Devuelve true en caso del éxito, de lo contrario devuelve false.

Nota

No se puede confundir las órdenes, transacciones y posiciones. Cada transacción es el resultado de la ejecución de una orden, mientras que cada posición es el resultado final de una o más transacciones.

La función HistoryDealSelect() limpia en un programa mql5 la lista de transacciones, las que están disponible, y copia en esta lista sólo una transacción, si la ejecución de la función HistoryDealSelect () se ha completado con éxito. Si hace falta repasar todas las transacciones seleccionadas por la función HistorySelect() es mejor utilizar la función HistoryDealGetTicket().

Véase también

HistorySelect(), HistoryDealGetTicket(), Propiedades de transacciones

HistoryDealsTotal

Devuelve el número de transacciones en el historial.

```
int  HistoryDealsTotal();
```

Valor devuelto

Valor del tipo int.

Nota

No se puede confundir las órdenes, transacciones y posiciones. Cada transacción es el resultado de la ejecución de una orden, mientras que cada posición es el resultado final de una o más transacciones.

Véase también

HistorySelect(), HistoryDealGetTicket(), Propiedades de transacciones

HistoryDealGetTicket

Elige una transacción a procesar posteriormente y devuelve el ticket de transacción en el historial.

```
ulong  HistoryDealGetTicket(
   int  index      // número de transacción
   );
```

Parámetros

index

[in] Número de transacción en la lista de transacciones.

Valor devuelto

Valor del tipo ulong. En caso de ejecución fallida devuelve 0.

Nota

No se puede confundir las órdenes, transacciones y posiciones. Cada transacción es el resultado de la ejecución de una orden, mientras que cada posición es el resultado final de una o más transacciones.

Ejemplo:

```
void OnStart()
  {
   ulong deal_ticket;            // ticket de transacción
   ulong order_ticket;           // ticket de la orden según la cual ha sido realizad
   datetime transaction_time;    // hora de realizar la transacción
   long deal_type ;              // tipo de operación comercial
   long position_ID;             // identificador de la posición
   string deal_description;      // descripción de la operación
   double volume;                // volumen de operación
   string symbol;                // símbolo usado en la transacción
//--- vamos a fijar la fecha inicial y final para solicitar el historial de transacci
   datetime from_date=0;          // desde el principio
   datetime to_date=TimeCurrent();// hasta el momento actual
//--- vamos a solicitar el historial de transacciones del período especificado
   HistorySelect(from_date,to_date);
//--- número total en la lista de transacciones
   int deals=HistoryDealsTotal();
//--- ahora vamos a procesar cada transacción
   for(int i=0;i<deals;i++)
     {
      deal_ticket=               HistoryDealGetTicket(i);
      volume=                    HistoryDealGetDouble(deal_ticket,DEAL_VOLUME);
      transaction_time=(datetime)HistoryDealGetInteger(deal_ticket,DEAL_TIME);
      order_ticket=              HistoryDealGetInteger(deal_ticket,DEAL_ORDER);
      deal_type=                 HistoryDealGetInteger(deal_ticket,DEAL_TYPE);
      symbol=                    HistoryDealGetString(deal_ticket,DEAL_SYMBOL);
      position_ID=               HistoryDealGetInteger(deal_ticket,DEAL_POSITION_ID);
      deal_description=          GetDealDescription(deal_type,volume,symbol,order_tic
      //--- hagamos el formateado bonito para el número de transacción
      string print_index=StringFormat("% 3d",i);
      //--- mostramos la información sobre la transacción
      Print(print_index+": deal #",deal_ticket," at ",transaction_time,deal_descripti
     }
  }
//+------------------------------------------------------------------+
//|  devuelve la descripción literal de la transacción               |
//+------------------------------------------------------------------+
string GetDealDescription(long deal_type,double volume,string symbol,long ticket,long
  {
   string descr;
//---
   switch(deal_type)
     {
      case DEAL_TYPE_BALANCE:                  return ("balance");
      case DEAL_TYPE_CREDIT:                   return ("credit");
      case DEAL_TYPE_CHARGE:                   return ("charge");
      case DEAL_TYPE_CORRECTION:               return ("correction");
      case DEAL_TYPE_BUY:                      descr="buy"; break;
      case DEAL_TYPE_SELL:                     descr="sell"; break;
      case DEAL_TYPE_BONUS:                    return ("bonus");
      case DEAL_TYPE_COMMISSION:               return ("additional commission");
      case DEAL_TYPE_COMMISSION_DAILY:         return ("daily commission");
      case DEAL_TYPE_COMMISSION_MONTHLY:       return ("monthly commission");
      case DEAL_TYPE_COMMISSION_AGENT_DAILY:   return ("daily agent commission");
      case DEAL_TYPE_COMMISSION_AGENT_MONTHLY: return ("monthly agent commission");
      case DEAL_TYPE_INTEREST:                 return ("interest rate");
      case DEAL_TYPE_BUY_CANCELED:             descr="cancelled buy deal"; break;
      case DEAL_TYPE_SELL_CANCELED:            descr="cancelled sell deal"; break;
     }
   descr=StringFormat("%s %G %s (order #%d, position ID %d)",
                      descr,  // descripción corriente
```

```
                        volume, // volumen de transacción
                        symbol, // instrumento de transacción
                        ticket, // ticket de la orden que ha provocado la transacción
                        pos_ID  // ID de la posición en la que ha participado la transa
                        );
   return(descr);
//---
  }
```

Véase también

HistorySelect(), HistoryDealsTotal(), HistoryDealSelect(), Propiedades de transacciones

HistoryDealGetDouble

La función devuelve la propiedad solicitada de una transacción. La propiedad de la transacción tiene que ser del tipo double. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
double  HistoryDealGetDouble(
   ulong                      ticket_number,     // ticket
   ENUM_DEAL_PROPERTY_DOUBLE  property_id        // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  HistoryDealGetDouble(
   ulong                      ticket_number,     // ticket
   ENUM_DEAL_PROPERTY_DOUBLE  property_id,       // identificador de la propiedad
   double&                    double_var         // aquí recibimos el valor de la pro
   );
```

Parámetros

ticket_number

[in] Ticket de transacción.

property_id

[in] Identificador de la propiedad de transacción. Su valor puede ser uno de los valores de la enumeración ENUM_DEAL_PROPERTY_DOUBLE.

double_var

[out] Variable del tipo double que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo double.

Nota

No se puede confundir las órdenes, transacciones y posiciones. Cada transacción es el resultado de la ejecución de una orden, mientras que cada posición es el resultado final de una o más transacciones.

Véase también

HistoryDealsTotal(), HistorySelect(), HistoryDealGetTicket(), Propiedades de transacciones

HistoryDealGetInteger

La función devuelve la propiedad solicitada de una transacción. La propiedad de la transacción tiene que ser del tipo datetime, int. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
long  HistoryDealGetInteger(
   ulong                      ticket_number,     // ticket
   ENUM_DEAL_PROPERTY_INTEGER  property_id       // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  HistoryDealGetInteger(
   ulong                      ticket_number,     // ticket
   ENUM_DEAL_PROPERTY_INTEGER  property_id,      // identificador de la propiedad
   long&                      long_var           // aquí recibimos el valor de la pr
   );
```

Parámetros

ticket_number

[in] Ticket de transacción.

property_id

[in] Identificador de la propiedad de transacción. Su valor puede ser uno de los valores de la enumeración ENUM_DEAL_PROPERTY_INTEGER.

long_var

[out] Variable del tipo long que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo long.

Nota

No se puede confundir las órdenes, transacciones y posiciones. Cada transacción es el resultado de la ejecución de una orden, mientras que cada posición es el resultado final de una o más transacciones.

Ejemplo:

```
//+------------------------------------------------------------------+
//| Trade function                                                   |
//+------------------------------------------------------------------+
void OnTrade()
  {
//--- obtenemos el ticket de la última transacción desde el historial de trading sema
   ulong last_deal=GetLastDealTicket();
   if(HistoryDealSelect(last_deal))
     {
      //--- tiempo de ejecución de transacción en milisegundos desde 01.01.1970
      long deal_time_msc=HistoryDealGetInteger(last_deal,DEAL_TIME_MSC);
      PrintFormat("Deal #%d DEAL_TIME_MSC=%i64 => %s",
                  last_deal,deal_time_msc,TimeToString(deal_time_msc/1000));
     }
   else
      PrintFormat("HistoryDealSelect() failed for #%d. Eror code=%d",
                  last_deal,GetLastError());
//---
  }
//+------------------------------------------------------------------+
//|  devuelve el ticket de la última transacción en el historial o -1              |
//+------------------------------------------------------------------+
ulong GetLastDealTicket()
  {
//--- solicitamos el historial de los últimos 7 días
   if(!GetTradeHistory(7))
     {
      //--- avisamos sobre la llamada fallida y devolvemos -1
      Print(__FUNCTION__," HistorySelect() ha devuelto false");
      return -1;
     }
//---
   ulong first_deal,last_deal,deals=HistoryOrdersTotal();
//--- si hay órdenes, empezamos a trabajar con ellas
   if(deals>0)
     {
      Print("Deals = ",deals);
      first_deal=HistoryDealGetTicket(0);
      PrintFormat("first_deal = %d",first_deal);
      if(deals>1)
        {
         last_deal=HistoryDealGetTicket((int)deals-1);
         PrintFormat("last_deal = %d",last_deal);
         return last_deal;
        }
      return first_deal;
     }
//--- no hemos encontrado ninguna transacción, devolveremos -1
   return -1;
  }
//+------------------------------------------------------------------+
//| solicita el historial de los últimos días y devuelve false en caso de fallo  |
//+------------------------------------------------------------------+
bool GetTradeHistory(int days)
  {
//--- fijamos un período de tiempo semanal para solicitar el historial de trading
   datetime to=TimeCurrent();
   datetime from=to-days*PeriodSeconds(PERIOD_D1);
   ResetLastError();
//--- hacemos la solicitud y comprobamos el resultado
   if(!HistorySelect(from,to))
```

```
      {
       Print(__FUNCTION__," HistorySelect=false. Error code=",GetLastError());
       return false;
      }
//--- historial recibido con éxito
   return true;
  }
```

Véase también

HistoryDealsTotal(), HistorySelect(), HistoryDealGetTicket(), Propiedades de transacciones

HistoryDealGetString

La función devuelve la propiedad solicitada de una transacción. La propiedad de la transacción tiene que ser del tipo string. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
string  HistoryDealGetString(
   ulong                      ticket_number,     // ticket
   ENUM_DEAL_PROPERTY_STRING  property_id        // identificador de la propiedad
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de la propiedad se coloca en una variable receptora que el último parámetro pasa por referencia.

```
bool  HistoryDealGetString(
   ulong                      ticket_number,     // ticket
   ENUM_DEAL_PROPERTY_STRING  property_id,       // identificador de la propiedad
   string&                    string_var         // aquí recibimos el valor de la pro
   );
```

Parámetros

ticket_number

[in] Ticket de transacción.

property_id

[in] Identificador de la propiedad de transacción. Su valor puede ser uno de los valores de la enumeración ENUM_DEAL_PROPERTY_STRING.

string_var

[out] Variable del tipo string que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo string.

Nota

No se puede confundir las órdenes, transacciones y posiciones. Cada transacción es el resultado de la ejecución de una orden, mientras que cada posición es el resultado final de una o más transacciones.

Véase también

HistoryDealsTotal(), HistorySelect(), HistoryDealGetTicket(), Propiedades de transacciones

Variables globales del terminal de cliente

Conjunto de funciones para trabajar con variables globales.

No se puede confundir las variables globales del terminal de cliente con las variables declaradas a nivel global del programa mql5.

Las variables globales se guardan en el terminal de cliente durante 4 semanas desde el último acceso, luego se eliminan automáticamente. El acceso a una variable global no es sólo la difinición de un valor nuevo, sino también la lectura del valor de una variable global.

Las variables globales del terminal de cliente están disponibles al mismo tiempo desde todos los programas mql5 abiertas en el terminal de cliente.

Función	Acción
GlobalVariableCheck	Comprueba la existencia de una variable global con el nombre especificado
GlobalVariableTime	Devuelve la hora del último acceso a una variable global
GlobalVariableDel	Elimina una variable global
GlobalVariableGet	Solicita el valor de una variable global
GlobalVariableName	Devuelve el nombre de una variable global según su número ordinal en la lista de variables globales
GlobalVariableSet	Establece el nuevo valor para una variable global
GlobalVariablesFlush	Guarda por vía forzada el contenido de todas las variables globales en el disco
GlobalVariableTemp	Establece el nuevo valor para una variable global que existe sólo durante esta sesión del terminal
GlobalVariableSetOnCondition	Establece el nuevo valor de una variable global ya existente según una condición
GlobalVariablesDeleteAll	Elimina variables globales con el prefijo especificado en su nombre
GlobalVariablesTotal	Devuelve la cantidad total de variables globales

GlobalVariableCheck

Comprueba la existencia de una variable global del terminal de cliente.

```
bool  GlobalVariableCheck(
   string  name      // nombre
   );
```

Parámetros

name

[in] Nombre de la variable global.

Valor devuelto

Devuelve el valor true, si la variable global existe, de lo contrario devuelve false.

Nota

Las variables globales se guardan en el terminal de cliente durante 4 semanas desde el último acceso, luego se eliminan automáticamente.

GlobalVariableTime

Devuelve la hora del último acceso a una variable global.

```
datetime  GlobalVariableTime(
   string  name      // nombre
   );
```

Parámetros

name

[in] Nombre de la variable global.

Valor devuelto

Devuelve la hora del último acceso a una variable global especificada. Llamar a una variable para conseguir el valor también es un acceso. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Las variables globales se guardan en el terminal de cliente durante 4 semanas desde el último acceso, luego se eliminan automáticamente.

Véase también

GlobalVariableCheck()

GlobalVariableDel

Elimina una variable global del terminal de cliente.

```
bool  GlobalVariableDel(
   string  name      // nombre
   );
```

Parámetros

name

[in] Nombre de la variable global.

Valor devuelto

Si se elimina con éxito, la función devuelve true, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Las variables globales se guardan en el terminal de cliente durante 4 semanas desde el último acceso, luego se eliminan automáticamente.

GlobalVariableGet

Devuelve el valor de una variable global ya existente del terminal de cliente. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
double  GlobalVariableGet(
   string  name      // nombre
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso del éxito el valor de una variable global se coloca en una variable receptora que es pasada por referencia por el segundo parámetro.

```
bool  GlobalVariableGet(
   string  name               // nombre
   double& double_var         // aquí recibimos el valor de la variable global
   );
```

Parámetros

name

[in] Nombre de la variable global.

double_var

[out] Variable del tipo double que recibe el valor guardado en una variable global del terminal de cliente.

Valor devuelto

El valor de la variable global existente o 0 en caso del error. Para obtener la información sobre el error hay que llamar a la función GetLastError().

Nota

Las variables globales se guardan en el terminal de cliente durante 4 semanas desde el último acceso, luego se eliminan automáticamente.

GlobalVariableName

Devuelve el nombre de una variable global según su número ordinal.

```
string  GlobalVariableName(
   int  index      // número en la lista de variables globales
   );
```

Parámetros

index

[in] Número ordinal en la lista de variables globales. Tiene que ser más de o igual a 0 y menos de GlobalVariablesTotal() .

Valor devuelto

Nombre de una variable global según su número ordinal en la lista de variables globales. Para obtener la información sobre el error hay que llamar a la función GetLastError().

Nota

Las variables globales se guardan en el terminal de cliente durante 4 semanas desde el último acceso, luego se eliminan automáticamente.

GlobalVariableSet

Establece el nuevo valor para una variable global. Si la variable no existe, el sistema creará una nueva variable global.

```
datetime  GlobalVariableSet(
   string  name,      // nombre
   double  value      // valor que se asigna
   );
```

Parámetros

name

[in] Nombre de la variable global.

value

[in] Valor numérico nuevo.

Valor devuelto

Si se ejecuta con éxito la función devuelve la hora del último acceso, de lo contrario devuelve 0. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Las variables globales se guardan en el terminal de cliente durante 4 semanas desde el último acceso, luego se eliminan automáticamente.

GlobalVariablesFlush

Guarda por vía forzada el contenido de todas las variables globales en el disco.

```
void  GlobalVariablesFlush();
```

Valor devuelto

No hay valor devuelto.

Nota

El terminal guarda todas las variables globales durante la finalización de su trabajo pero con un fallo repentino del ordenador los datos pueden perderse. Esta función permite dirigir el proceso de guardado de variables globales de una manera independiente en caso de una emergencia.

GlobalVariableTemp

Intenta crear una nueva variable global. Si la variable no existe, el sistema creará una nueva variable global temporal.

```
bool  GlobalVariableTemp(
   string  name,      // nombre
   );
```

Parámetros

name

[in] Nombre de la variable global temporal.

Valor devuelto

En caso de éxito la función devuelve true, de lo contrario devuelve false. Para obtener la información sobre el error hay que llamar a la función GetLastError().

Nota

Las variables globales temporales existen sólo durante el funcionamiento del terminal de cliente, después del cierre del terminal estas variables se eliminan automáticamente. Durante la ejecución de la operación GlobalVariablesFlush() las variables globales temporales no se guardan en el disco.

Una vez creada una variable global temporal, el acceso a ella y su modificación se realizan igual que a una variable global del terminal de cliente común.

GlobalVariableSetOnCondition

Establece un nuevo valor de una variable global existente, si el valor actual de la variable es igual al valor del tercer parámetro check_value. Si la variable no existe, la función generará el error ERR_GLOBALVARIABLE_NOT_FOUND (4501) y devolverá false.

```
bool  GlobalVariableSetOnCondition(
   string  name,            // nombre
   double  value,           // valor si se cumple la condición
   double  check_value      // condición que se comprueba
   );
```

Parámetros

name

[in] Nombre de la variable global.

value

[in] Valor nuevo.

check_value

[in] Valor para comprobar el valor actual de la variable global.

Valor devuelto

En caso de éxito la función devuelve true, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError(). Si el valor actual de la variable global es diferente a check_value, la función devolverá false.

Nota

La función proporciona el acceso atómico a una variable global, por eso se puede usarla para organizar un mutex en caso de interacción de varios Asesores Expertos que trabajan al mismo tiempo dentro de un terminal de cliente.

GlobalVariablesDeleteAll

Elimina variables globales del terminal de cliente.

```
int  GlobalVariablesDeleteAll(
   string     prefix_name=NULL      // todas las variables globales cuyos nombres se
   datetime   limit_data=0          // todas las variables globales que han sido camb
   );
```

Parámetros

prefix_name=NULL

[in] Prefijo del nombre de las variables globales que se eliminan. Si se trata del prefijo NULL o cadena vacía, entonces todas las variables globales que cumplen con el criterio de la fecha serán eliminadas.

limit_data=0

[in] Fecha para la selección de variables globales según el momento de su última modificación. Las variables globales que se han modificado antes de la fecha indicada serán eliminadas. Si el parámetro es igual a cero, se eliminan todas las variables globales que cumplen con el primer criterio (según el prefijo).

Valor devuelto

Número de variables eliminadas.

Nota

Si ambos parámetros son iguales a cero (prefix_name=NULL y limit_data=0), se eliminan todas las variables globales del terminal. Si los dos parámetros están especificados, se eliminan las variables globales correspondientes a cada uno de los parámetros especificados.

Las variables globales se guardan en el terminal de cliente durante 4 semanas desde el último acceso, luego se eliminan automáticamente.

GlobalVariablesTotal

Devuelve la cantidad total de variables globales del terminal de cliente.

```
int  GlobalVariablesTotal();
```

Valor devuelto

Número de variables globales.

Nota

Las variables globales se guardan en el terminal de cliente durante 4 semanas desde el último acceso, luego se eliminan automáticamente. El acceso a una variable global no es sólo la difinición de un valor nuevo, sino también la lectura del valor de una variable global.

Operaciones con archivos

Éste es el grupo de funciones que se utilizan para operar con los archivos.

Existen dos carpetas (con subcarpetas) en las que se puede colocar los archivos de trabajo:

- terminal_data_directorio\MQL5\FILES\ (para verla seleccione el punto del menú "Archivo"-"Abrir carpeta de datos" en el terminal);
- carpeta general de todos los terminales instalados en el ordenador - suele ubicarse en el directorio C:\Documents and Settings\All Users\Application Data\MetaQuotes\Terminal\Common\.

Se puede obtener los nombres de estos catálogos de forma de programación empleando la función TerminalInfoString() y usando las enumeraciones ENUM_TERMINAL_INFO_STRING:

```
//--- Carpeta en la que se guardan los datos del terminal
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
//--- Directorio general de todos los terminales de cliente
   string common_data_path=TerminalInfoString(TERMINAL_COMMONDATA_PATH);
```

Está prohibido trabajar con los archivos desde otras carpetas.

Las funciones de archivos permiten trabajar con lo que llaman "tuberías nombradas". Para eso sólo hay que llamar a la función FileOpen() con los parámetros correspondientes.

Función	Acción
FileFindFirst	Empieza la búsqueda de los archivos en el directorio correspondiente de acuerdo con el filtro especificado
FileFindNext	Sigue con la búsqueda empezada por la función FileFindFirst()
FileFindClose	Cierra el manejador de búsqueda
FileOpen	Abre un archivo con el nombre y banderas especificados
FileDelete	Elimina un archivo especificado
FileFlush	Guarda en el disco todos los datos que se han quedado en el buffer de entrada/salida
FileGetInteger	Obtiene una propiedad del número entero del archivo
FileIsEnding	Determina el final de un archivo en el proceso de lectura
FileIsLineEnding	Determina el fin de una línea en un archivo de texto en el proceso de lectura
FileClose	Cierra un archivo previamente abierto
FileIsExist	Comprueba la existencia de un archivo
FileCopy	Copia el archivo original de una carpeta local o

	compartida a otro archivo
FileMove	Mueve o renombra un archivo
FileReadArray	Lee los arrays de cualquier tipo, salvo los arrays literales (string) (puede ser un array de estructuras que no contienen las cadenas ni arrays dinámicos) de un archivo binario desde la posición actual del puntero de archivos
FileReadBool	Lee de un archivo del tipo CSV una cadena desde la posición actual hasta el separador (o hasta el final de la línea de texto) y convierte la cadena leída al valor del tipo bool
FileReadDatetime	Lee de un archivo del tipo CSV una cadena de uno de los formatos: "YYYY.MM.DD HH:MI:SS", "YYYY.MM.DD" o "HH:MI:SS" - y la convierte al valor del tipo datetime
FileReadDouble	Lee un número de doble precisión con punto flotante (double) de un archivo binario desde la posición actual del puntero de archivos
FileReadFloat	Lee desde la posición actual del puntero de archivos valor del tipo float
FileReadInteger	Lee de un archivo binario valor del tipo int, short o char dependiendo de la longitud indicada en bytes
FileReadLong	Lee desde la posición actual del puntero de archivos valor del tipo long
FileReadNumber	Lee de un archivo del tipo CSV una cadena desde la posición actual hasta el separador (o hasta el final de la línea de texto) y convierte la cadena leída al valor del tipo double
FileReadString	Lee de un archivo una cadena desde la posición actual del puntero de archivos
FileReadStruct	Lee de un archivo binario el contenido en una estructura que ha sido pasada como un parámetro
FileSeek	Mueve la posición del puntero de archivos a una cantidad de bytes especificada respecto a la posición indicada
FileSize	Devuelve el tamaño de un archivo correspondiente abierto
FileTell	Devuelve la posición actual del puntero de archivos de un archivo correspondiente abierto
FileWrite	Escribe los datos en un archivo del tipo CSV o

	TXT
FileWriteArray	Escribe los arrays de cualquier tipo (excepto los arrays string) en un archivo del tipo BIN
FileWriteDouble	Escribe el valor del parámetro del tipo double desde la posición actual del puntero de archivos en un archivo binario
FileWriteFloat	Escribe el valor del parámetro del tipo float desde la posición actual del puntero de archivos en un archivo binario
FileWriteInteger	Escribe el valor del parámetro del tipo int desde la posición actual del puntero de archivos en un archivo binario
FileWriteLong	Escribe el valor del parámetro del tipo long desde la posición actual del puntero de archivos en un archivo binario
FileWriteString	Escribe el valor del parámetro del tipo string desde la posición actual del puntero de archivos en un archivo del tipo BIN o TXT
FileWriteStruct	Escribe el contenido de una estructura pasada como un parámetro en un archivo binario desde la posición actual del puntero de archivos
FolderCreate	Crea un directorio en la carpeta Files (dependiendo del valor common_flag)
FolderDelete	Elimina un directorio seleccionado. Una carpeta no vacía no puede ser eliminada
FolderClean	Elimina todos los archivos en la carpeta especificada

Si el archivo se abre para escritura usando la función FileOpen(), todas las subcarpetas indicadas en la ruta van a ser creadas en caso si no existen.

FileFindFirst

La función empieza la búsqueda de los archivos y subcarpetas en el directorio correspondiente de acuerdo con el filtro especificado.

```
long  FileFindFirst(
   const string   file_filter,          // cadena - filtro de búsqueda
   string&        returned_filename,    // nombre del archivo o subcarpeta encontrada
   int            common_flag=0         // determina la zona de búsqueda
   );
```

Parámetros

file_filter

[in] Filtro de búsqueda. En el filtro se puede especificar un subdirectorio (o una sucesión de subdirectorios anidados) respecto al directorio \Files en el que se precisa realizar la búsqueda.

returned_filename

[out] Parámetro devuelto en el que se coloca el nombre del primer archivo o subcarpeta encontrada en caso del éxito.

common_flag

[in] Bandera que determina la ubicación del archivo. Si common_flag=FILE_COMMON, entonces el archivo se encuentra en una carpeta compartida de todos los terminales de cliente. De lo contrario, el archivo está ubicado en una carpeta local.

Valor devuelto

Devuelve el manejador del objeto de búsqueda que hay que utilizar para la siguiente búsqueda de archivos y subcarpetas por la función FileFindNext(), o INVALID_HANDLE en caso no hay ningún archivo o subcarpeta que corresponda al filtro (en caso particular - la subcarpeta está vacía). Después de finalizar la búsqueda, hay que cerrar el manejador usando la función FileFindClose().

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- filtro
input string InpFilter="*";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   int    i=1;
//--- obtenemos el manejador de búsqueda en la raíz de la carpeta local
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- comprobamos si la función FileFindFirst() se ha ejecutado con éxito
   if(search_handle!=INVALID_HANDLE)
     {
      //--- comprobamos en el ciclo si las cadenas pasadas son los nombres de los arc
      do
        {
         ResetLastError();
         //--- si es un archivo, la función devuelve true, y si es una carpeta, la fu
         FileIsExist(file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==5018 ? "Directory" : "File
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- cerramos el manejador de la búsqueda
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

Véase también

FileFindNext, FileFindClose

FileFindNext

Sigue con la búsqueda empezada por la función FileFindFirst().

```
bool  FileFindNext(
   long      search_handle,          // manejador de búsqueda
   string&   returned_filename       // nombre del archivo o subcarpeta encontrada
   );
```

Parámetros

search_handle

[in] Manejador de búsqueda recibido de la función FileFindFirst().

returned_filename

[out] Nombre del siguiente archivo o subcarpeta encontrada.

Valor devuelto

En caso del éxito devuelve true, de lo contrario devuelve false.

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- filtro
input string InpFilter="*";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   int    i=1;
//--- obtenemos el manejador de búsqueda en la raíz de la carpeta local
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- comprobamos si la función FileFindFirst() se ha ejecutado con éxito
   if(search_handle!=INVALID_HANDLE)
     {
      //--- comprobamos en el ciclo si las cadenas pasadas son los nombres de los arc
      do
        {
         ResetLastError();
         //--- si es un archivo, la función devuelve true, y si es una carpeta, la fu
         FileIsExist(file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==5018 ? "Directory" : "File
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- cerramos el manejador de la búsqueda
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

Véase también

FileFindFirst, FileFindClose

FileFindClose

Cierra el manejador de búsqueda.

```
void  FileFindClose(
   long  search_handle      // manejador de búsqueda
   );
```

Parámetros

search_handle

[in] Manejador de búsqueda recibido de la función FileFindFirst().

Valor devuelto

No hay valor devuelto.

Nota

Hay que llamar a la función para liberar los recursos del sistema.

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- filtro
input string InpFilter="*";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string file_name;
   int    i=1;
//--- obtenemos el manejador de búsqueda en la raíz de la carpeta local
   long search_handle=FileFindFirst(InpFilter,file_name);
//--- comprobamos si la función FileFindFirst() se ha ejecutado con éxito
   if(search_handle!=INVALID_HANDLE)
     {
      //--- comprobamos en el ciclo si las cadenas pasadas son los nombres de los arc
      do
        {
         ResetLastError();
         //--- si es un archivo, la función devuelve true, y si es una carpeta, la fu
         FileIsExist(file_name);
         PrintFormat("%d : %s name = %s",i,GetLastError()==5018 ? "Directory" : "File
         i++;
        }
      while(FileFindNext(search_handle,file_name));
      //--- cerramos el manejador de la búsqueda
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
  }
```

Véase también

FileFindFirst, FileFindNext

FileIsExist

Esta función comprueba la existencia de un archivo.

```
bool  FileIsExist(
   const string  file_name,        // nombre del archivo
   int           common_flag=0     // zona de búsqueda
   );
```

Parámetros

file_name

[in] Nombre del archivo a comprobar.

common_flag=0

[in] Bandera que determina la ubicación del archivo. Si common_flag=FILE_COMMON, entonces el archivo se encuentra en una carpeta compartida de todos los terminales de cliente. De lo contrario, el archivo está ubicado en una carpeta local.

Valor devuelto

Devuelve true, si el archivo especificado existe.

Nota

El archivo comprobado puede resultar una subcarpeta. En este caso la función FileIsExist() devuelve false y en la variable _LastError se inscribe el error 5018 - "No es un archivo, sino una subcarpeta" (ver el ejemplo para la función FileFindFirst).

Por razones de seguridad el trabajo con los archivos en el lenguaje MQL5 se encuentra bajo un estricto control. Los archivos con los que se realizan las operaciones de archivos utilizando los medios del lenguaje MQL5 no pueden estar fuera del entorno protegido de archivos (file sandbox).

Si common_flag=FILE_COMMON, la función busca el archivo especificado en una carpeta compartida de todos los terminales de cliente, de lo contrario la función lo busca en una carpeta local (MQL5\Files o MQL5\Tester\Files en caso de prueba).

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- fecha para archivos antiguos
input datetime InpFilesDate=D'2013.01.01 00:00';
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // variable para almacenar los nombres de archivos
   string   filter="*.txt"; // filtro para la búsqueda de archivos
   datetime create_date;    // fecha de creación del archivo
   string   files[];        // lista con los nombres de los archivos
   int      def_size=25;    // tamaño del array por defecto
   int      size=0;         // número de archivos
//--- adjudicar memoria para array
   ArrayResize(files,def_size);
//--- recibir el manejador de búsqueda en la raíz de la carpeta local
   long search_handle=FileFindFirst(filter,file_name);
//--- comprobamos si la función FileFindFirst() se ha ejecutado con exito
   if(search_handle!=INVALID_HANDLE)
     {
      //--- repasamos los archivos en el ciclo
      do
        {
         files[size]=file_name;
         //--- aumentamos el tamaño del array
         size++;
         if(size==def_size)
           {
            def_size+=25;
            ArrayResize(files,def_size);
           }
         //--- reseteamos el valor del error
         ResetLastError();
         //--- obtenemos la fecha de creación del archivo
         create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
         //--- comprobamos si el archivo es antiguo
         if(create_date<InpFilesDate)
           {
            PrintFormat("¡El archivo %s ha sido eliminado!",file_name);
            //--- eliminamos el archivo antiguo
            FileDelete(file_name);
           }
        }
      while(FileFindNext(search_handle,file_name));
      //--- cerramos el manejador de búsqueda
      FileFindClose(search_handle);
     }
   else
     {
      Print("Files not found!");
      return;
     }
//--- comprobamos qué archivos se han quedado
   PrintFormat("Resultados:");
   for(int i=0;i<size;i++)
     {
      if(FileIsExist(files[i]))
         PrintFormat("¡El archivo %s existe!",files[i]);
      else
```

```
         PrintFormat("¡El archivo %s ha sido eliminado!",files[i]);
      }
   }
```

Véase también

FileFindFirst

FileOpen

La función abre el archivo con el nombre especificado y las banderas especificadas.

```
int  FileOpen(
   string  file_name,           // nombre del archivo
   int     open_flags,          // combinación de banderas
   short   delimiter='\t',      // delimitador
   uint    codepage=CP_ACP      // página de código
   );
```

Parámetros

file_name

[in] Nombre del archivo a abrir, puede contener subcarpetas. Si el archivo se abre para la escritura, las subcarpetas especificadas serán creadas en caso de que no existan.

open_flags

[in] combinación de banderas que determina el modo de trabajo con el archivo. Las banderas están definidas como sigue:
FILE_READ el archivo se abre para la lectura
FILE_WRITE el archivo se abre para la escritura
FILE_BIN modo binario de lectura-escritura (sin conversión de una cadena, ni tampoco en una cadena)
FILE_CSV archivo del tipo csv (todos los elementos grabados se convierten a las cadenas del tipo correspondiente, unicode o ansi, y se separan con un delimitador)
FILE_TXT archivo de texto simple (igual que el archivo csv pero sin tomar en cuenta los delimitadores)
FILE_ANSI cadenas del tipo ANSI (símbolos de un byte)
FILE_UNICODE cadenas del tipo UNICODE (símbolos de dos bytes)
FILE_SHARE_READ acceso compartido de lectura de parte de varios programas
FILE_SHARE_WRITE acceso compartido de escritura de parte de varios programas
FILE_COMMON ubicación del archivo en una carpeta compartida de todos los terminales de cliente

delimiter='\t'

[in] valor que se usa como un separador en el archivo txt o csv. Si para el archivo csv el delimitador no está especificado, por defecto se emplea el símbolo de tabulación. Si para el archivo txt el delimitador no está especificado, no se usa ningún separador. Si el valor 0 está establecido como un separador, no se utiliza ningún separador.

codepage=CP_ACP

[in] Valor de la página de código. Están previstas las constantes correspondientes para las páginas de códigos más usadas.

Valor devuelto

En caso de abrir con éxito, la función devuelve el manejador del archivo que luego se usa para acceder a los datos del archivo. En caso de fallo devuelve INVALID_HANDLE.

Nota

Por razones de seguridad el trabajo con los archivos en el lenguaje MQL5 se encuentra bajo un estricto control. Los archivos con los que se realizan las operaciones de archivos utilizando los

medios del lenguaje MQL5 no pueden estar fuera del entorno protegido de archivos (file sandbox).

El archivo se abre en la carpeta del terminal de cliente en la subcarpeta MQL5\files (o catálogo_del_agente_de_simulación\MQL5\files en caso de prueba). Si entre las banderas está indicada FILE_COMMON, el archivo se abre en la carpeta compartida de todos los terminales de cliente de MetaTrader5.

"Las tuberías nombradas" pueden ser abiertas de acuerdo con las siguientes reglas:

- El nombre de la tubería es la cadena que debe tener la siguiente apariencia: "\\servername\pipe\pipename", donde servername es el nombre del servidor en la red y pipename es el nombre de la tubería. Si las tuberías se utilizan en el mismo ordenador, se puede omitir el nombre del servidor, pero a su vez hay que poner un punto: "\\.\pipe\pipename". El cliente que intenta conectarse con la tubería tiene que saber su nombre.
- Hay que llamar a las funciones FileFlush() y FileSeek() al inicio del archivo entre las operaciones consecutivas de lectura desde la tubería y la escritura en ella.

En las cadenas mostradas se utiliza un símbolo especial, la barra inversa '\'. Por eso, cuando se escribe el nombre en el programa MQL4, hace falta duplicar la barra inversa '\'. Es decir, escribir el ejemplo arriba mencionado en el código de la siguiente manera: "\\\\servername\\pipe\\pipename".

Puede encontrar más información sobre el trabajo con las tuberías nombradas en el artículo "Communicating With MetaTrader 5 Using Named Pipes Without Using DLLs".

Ejemplo:

```
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- modo incorrecto de abrir un archivo
   string terminal_data_path=TerminalInfoString(TERMINAL_DATA_PATH);
   string filename=terminal_data_path+"\\MQL5\\Files\\"+"fractals.csv";
   int filehandle=FileOpen(filename,FILE_WRITE|FILE_CSV);
   if(filehandle<0)
     {
      Print("Fallo al abrir el archivo por la ruta absoluta");
      Print("Código de error ",GetLastError());
     }

//--- modo correcto de operar en la zona protegida de archivos
   ResetLastError();
   filehandle=FileOpen("fractals.csv",FILE_WRITE|FILE_CSV);
   if(filehandle!=INVALID_HANDLE)
     {
      FileWrite(filehandle,TimeCurrent(),Symbol(),PERIOD_CURRENT);
      FileClose(filehandle);
      Print("FileOpen OK");
     }
   else Print("Operación FileOpen fallida, error ",GetLastError());
//--- otro ejemplo más de crear un directorio anidado dentro MQL5\Files\
```

```
   string subfolder="Research";
   filehandle=FileOpen(subfolder+"\\fractals.txt",FILE_WRITE|FILE_CSV);
      if(filehandle!=INVALID_HANDLE)
     {
      FileWrite(filehandle,TimeCurrent(),Symbol(),PERIOD_CURRENT);
      FileClose(filehandle);
      Print("El archivo debe ser creado en la carpeta "+terminal_data_path+"\\"+subfo
     }
   else Print("File open failed, error ",GetLastError());
  }
```

Véase también

Uso de página de código, FileFindFirst, FolderCreate, Banderas de apertura de archivos

FileClose

Cierra el archivo previamente abierto por la función FileOpen().

```
void  FileClose(
   int  file_handle      // manejador de archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

No hay valor devuelto.

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros de entrada
input string InpFileName="file.txt";     // nombre del archivo
input string InpDirectoryName="Data";    // nombre de la carpeta
input int    InpEncodingType=FILE_ANSI; // ANSI=32 o UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- imprimimos la ruta de la carpeta en la que vamos a trabajar
   PrintFormat("Trabajamos en la carpeta %s\\Files\\",TerminalInfoString(TERMINAL_DAT
//--- reseteamos el valor del error
   ResetLastError();
//--- abrimos el archivo de lectura (si el archivo no existe, se produce un error)
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpE
   if(file_handle!=INVALID_HANDLE)
     {
      //--- imprimimos el contenido del archivo
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      //--- cerramos el archivo
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, código = %d",GetLastError());
  }
```

FileCopy

Copia el archivo original de una carpeta local o compartida a otro archivo.

```
bool  FileCopy(
   const string  src_file_name,     // nombre del archivo-fuente
   int           common_flag,       // zona de acción
   const string  dst_file_name,     // nombre del archivo de destino
   int           mode_flags         // modo de acceso
   );
```

Parámetros

src_file_name

[in] Nombre del archivo a copiar.

common_flag

[in] Bandera que determina la ubicación del archivo. Si common_flag=FILE_COMMON, entonces el archivo se encuentra en una carpeta compartida de todos los terminales de cliente. De lo contrario, el archivo está ubicado en una carpeta local (por ejemplo, *common_flag=0*).

dst_file_name

[in] Nombre del archivo resultante.

mode_flags

[in] Banderas de acceso. El parámetro puede contener sólo 2 banderas: FILE_REWRITE y/o FILE_COMMON - las demás banderas se ignoran. Si el archivo ya existe y la bandera FILE_REWRITE no ha sido especificada, el archivo no se regrabará, y la función devolverá false.

Valor devuelto

En caso de fallo la función devuelve false.

Nota

Si el archivo nuevo ya existía, el copiado va a realizarse dependiendo de la presencia de la bandera FILE_REWRITE en el parámetro mode_flags.

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros de entrada
input string InpSrc="source.txt";       // fuente
input string InpDst="destination.txt";  // copia
input int    InpEncodingType=FILE_ANSI; // ANSI=32 o UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- mostramos el contenido de la fuente (tiene que existir)
   if(!FileDisplay(InpSrc))
      return;
//--- comprobamos si existe ya el archivo de la copia (no es obligatorio que haya sid
   if(!FileDisplay(InpDst))
     {
      //--- el archivo de la copia no existe, el copiado sin bandera FILE_REWRITE (co
      if(FileCopy(InpSrc,0,InpDst,0))
         Print("File is copied!");
      else
         Print("File is not copied!");
     }
   else
     {
      //--- el archivo de la copia ya existe, intentemos copiar sin bandera FILE_REWR
      if(FileCopy(InpSrc,0,InpDst,0))
         Print("File is copied!");
      else
         Print("File is not copied!");
      //--- el contenido del archivo InpDst se queda el mismo
      FileDisplay(InpDst);
      //--- volvemos a copiar con la bandera FILE_REWRITE (copiado correcto cuando el
      if(FileCopy(InpSrc,0,InpDst,FILE_REWRITE))
         Print("File is copied!");
      else
         Print("File is not copied!");
     }
//--- recibida la copia del archivo InpSrc
   FileDisplay(InpDst);
  }
//+------------------------------------------------------------------+
//| Lectura del contenido de archivo                                 |
//+------------------------------------------------------------------+
bool FileDisplay(const string file_name)
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- abrimos el archivo
   int file_handle=FileOpen(file_name,FILE_READ|FILE_TXT|InpEncodingType);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- mostramos en el ciclo el contenido del archivo
      Print("+----------------------+");
      PrintFormat("File name = %s",file_name);
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      Print("+----------------------+");
      //--- cerramos archivo
      FileClose(file_handle);
      return(true);
```

```
      }
//--- no se puede abrir el archivo
   PrintFormat("%s is not opened, error = %d",file_name,GetLastError());
   return(false);
  }
```

FileDelete

Elimina el archivo especificado en una carpeta local del terminal de cliente.

```
bool  FileDelete(
   const string  file_name,     // nombre del archivo a eliminar
   int           common_flag=0  // ubicación del archivo a eliminar
   );
```

Parámetros

file_name

[in] Nombre del archivo.

common_flag=0

[in] Bandera que determina la ubicación del archivo. Si common_flag=FILE_COMMON, entonces el archivo se encuentra en una carpeta compartida de todos los terminales de cliente. De lo contrario, el archivo está ubicado en una carpeta local

Valor devuelto

En caso de fallo la función devuelve false.

Nota

Por razones de seguridad el trabajo con los archivos en el lenguaje MQL5 se encuentra bajo un estricto control. Los archivos con los que se realizan las operaciones de archivos utilizando los medios del lenguaje MQL5 no pueden estar fuera del entorno protegido de archivos (file sandbox).

Elimina el archivo especificado en una carpeta local del terminal de cliente (MQL5\files o MQL5 \tester\files en caso de prueba). Pero si hay common_flag=FILE_COMMON, la función elimina el archivo de la carpeta compartida de todos los terminales de cliente.

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- fecha para archivos antiguos
input datetime InpFilesDate=D'2013.01.01 00:00';
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // variable para almacenar los nombres de archivos
   string   filter="*.txt"; // filtro para la búsqueda de archivos
   datetime create_date;    // fecha de creación del archivo
   string   files[];        // lista con los nombres de los archivos
   int      def_size=25;    // tamaño del array por defecto
   int      size=0;         // número de archivos
//--- adjudicar memoria para array
   ArrayResize(files,def_size);
//--- recibir el manejador de búsqueda en la raíz de la carpeta local
   long search_handle=FileFindFirst(filter,file_name);
//--- comprobamos si la función FileFindFirst() se ha ejecutado con exito
   if(search_handle!=INVALID_HANDLE)
     {
      //--- repasamos los archivos en el ciclo
      do
        {
         files[size]=file_name;
         //--- aumentamos el tamaño del array
         size++;
         if(size==def_size)
           {
            def_size+=25;
            ArrayResize(files,def_size);
           }
         //--- reseteamos el valor del error
         ResetLastError();
         //--- obtenemos la fecha de creación del archivo
         create_date=(datetime)FileGetInteger(file_name,FILE_CREATE_DATE,false);
         //--- comprobamos si el archivo es antiguo
         if(create_date<InpFilesDate)
           {
            PrintFormat("¡El archivo %s ha sido eliminado!",file_name);
            //--- eliminamos el archivo antiguo
            FileDelete(file_name);
           }
        }
      while(FileFindNext(search_handle,file_name));
      //--- cerramos el manejador de búsqueda
      FileFindClose(search_handle);
     }
   else
     {
      Print("Files not found!");
      return;
     }
//--- comprobamos qué archivos se han quedado
   PrintFormat("Resultados:");
   for(int i=0;i<size;i++)
     {
      if(FileIsExist(files[i]))
         PrintFormat("¡El archivo %s existe!",files[i]);
      else
```

```
         PrintFormat("¡El archivo %s ha sido eliminado!",files[i]);
      }
   }
```

FileMove

Mueve un archivo de una carpeta local o compartida a otra carpeta.

```
bool  FileMove(
   const string  src_file_name,     // nombre del archivo para la operación de mover
   int           common_flag,       // zona de acción
   const string  dst_file_name,     // nombre del archivo de destino
   int           mode_flags         // modo de acceso
   );
```

Parámetros

src_file_name

[in] Nombre del archivo a mover/renombrar.

common_flag

[in] Bandera que determina la ubicación del archivo. Si common_flag=FILE_COMMON, entonces el archivo se encuentra en una carpeta compartida de todos los terminales de cliente. De lo contrario, el archivo está ubicado en una carpeta local (*common_flag=0*).

dst_file_name

[in] Nombre del archivo resultante.

mode_flags

[in] Banderas de acceso. El parámetro puede contener sólo 2 banderas: FILE_REWRITE y/o FILE_COMMON - las demás banderas se ignoran. Si el archivo ya existe y la bandera FILE_REWRITE no ha sido especificada, el archivo no se regrabará, y la función devolverá false.

Valor devuelto

En caso de fallo la función devuelve false.

Nota

Por razones de seguridad el trabajo con los archivos en el lenguaje MQL5 se encuentra bajo un estricto control. Los archivos con los que se realizan las operaciones de archivos utilizando los medios del lenguaje MQL5 no pueden estar fuera del entorno protegido de archivos (file sandbox).

Si el archivo nuevo ya existía, el copiado va a realizarse dependiendo de la presencia de la bandera FILE_REWRITE en el parámetro mode_flags.

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros de entrada
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);
//--- obtenemos las rutas de archivos
   string src_path;
   string dst_path;
   StringConcatenate(src_path,InpSrcDirectory,"//",InpSrcName);
   StringConcatenate(dst_path,InpDstDirectory,"//",InpDstName);
//--- comprobamos si existe el archivo de la fuente (si no, a salir)
   if(FileIsExist(src_path))
      PrintFormat("%s file exists in the %s\\Files\\%s folder",InpSrcName,local,InpSr
   else
     {
      PrintFormat("Error, %s source file not found",InpSrcName);
      return;
     }
//--- comprobamos si existe el archivo del resultado
   if(FileIsExist(dst_path,FILE_COMMON))
     {
      PrintFormat("%s file exists in the %s\\Files\\%s folder",InpDstName,common,InpD
      //--- el archivo existe, hay que realizar el traslado con la bandera FILE_REWRI
      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",InpDstName,com
      //--- el archivo no existe, el traslado se realiza sin la bandera FILE_REWRITE
      ResetLastError();
      if(FileMove(src_path,0,dst_path,FILE_COMMON))
         PrintFormat("%s file moved",InpSrcName);
      else
         PrintFormat("Error! Code = %d",GetLastError());
     }
//--- ahora el archivo ya ha sido movido, vamos a comprobarlo
   if(FileIsExist(dst_path,FILE_COMMON) && !FileIsExist(src_path,0))
      Print("Success!");
   else
      Print("Error!");
  }
```

Véase también

FileIsExist

FileFlush

Esta función guarda en el disco todos los datos que se han quedado en el búfer de entrada/salida.

```
void  FileFlush(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

No hay valor devuelto.

Nota

Cuando se ejecuta la operación de escrituro en el archivo, los datos físicos pueden aparecer en él sólo dentro de un período de tiempo. Para que los datos se guarden en el archivo en el acto, hay que utilizar la función FileFlush(). Si esta función no se utiliza, una parte de datos que todavía no han llegado al disco se graban en él forzadamente sólo cuando la función FileClose() cierra el archivo.

Es necesario utilizar esta función cuando los datos que se graban representan un cierto valor. Y hay que tener en cuenta que las llamadas frecuentes a esta función pueden reflejarse en la velocidad del programa.

Hay que llamar a la función FileFlush() entre las operaciones de lectura y escritura de un archivo.

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al arranciar el script
#property script_show_inputs
//--- nombre de archivo para la escritura
input string InpFileName="example.csv"; // nombre del archivo
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- reseteamos el valor del error
   ResetLastError();
//--- abrimos el archivo
   int file_handle=FileOpen(InpFileName,FILE_READ|FILE_WRITE|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- grabamos los datos en el archivo
      for(int i=0;i<1000;i++)
        {
         //--- llamamos la función de escritura
         FileWrite(file_handle,TimeCurrent(),SymbolInfoDouble(Symbol(),SYMBOL_BID),Sy
         //--- grabamos los datos en el disco con cada 128 iteración
         if((i & 127)==127)
           {
            //--- ahora los datos van a guardarse en el archivo, y en caso de un erro
            FileFlush(file_handle);
            PrintFormat("i = %d, OK",i);
           }
         //--- retraso de 0,01 segundos
         Sleep(10);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, código = %d",GetLastError());
  }
```

Véase también

FileClose

FileGetInteger

Recibe una propiedad de números enteros del archivo. Hay 2 variantes de esta función.

1. Obtención de propiedades por el manejador del archivo.

```
long  FileGetInteger(
   int                          file_handle,          // manejador del archivo
   ENUM_FILE_PROPERTY_INTEGER   property_id           // identificador de la propiedad
   );
```

2. Obtención de propiedades por el nombre del archivo.

```
long  FileGetInteger(
   const string                 file_name,            // nombre del archivo
   ENUM_FILE_PROPERTY_INTEGER   property_id,          // identificador de la propiedad
   bool                         common_folder=false   // el archivo se busca en una ca
   );                                                 // o en la carpeta común de todo
```

Parámetros

file_handle

[in] Descriptor de archivos devuelto por la función FileOpen().

file_name

[in] Nombre del archivo.

property_id

[in] Identificador de la propiedad del archivo. El valor puede ser uno de los valores de la enumeración ENUM_FILE_PROPERTY_INTEGER. Si se utiliza la segunda variante de la función, entonces se puede obtener los valores sólo de las siguientes propiedades: FILE_EXISTS, FILE_CREATE_DATE, FILE_MODIFY_DATE, FILE_ACCESS_DATE y FILE_SIZE.

common_folder=false

[in] Indica la ubicación del archivo. Si el parámetro es igual a false, entonces se revisa la carpeta de datos del terminal, en caso contrario se supone que el archivo se encuentra en la carpeta común de todos los terminales de cliente (FILE_COMMON).

Valor devuelto

Valor de la propiedad. En caso del error la función devuelve -1. Para obtener el código del error, se debe llamar a la función GetLastError().

Si durante la obtención de propiedades por el nombre se especifica una carpeta, en cualquier caso la función pondrá el error 5018 (ERR_MQL_FILE_IS_DIRECTORY), y el valor devuelto será correcto.

Nota

La función siempre cambia el código del error. En caso de una finalización con éxito el código del error se pone a cero.

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al arrancar el script
#property script_show_inputs
//--- parámetros de entrada
input string InpFileName="data.csv";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string path=InpDirectoryName+"//"+InpFileName;
   long   l=0;
//--- abrimos el archivo
   ResetLastError();
   int handle=FileOpen(path,FILE_READ|FILE_CSV);
   if(handle!=INVALID_HANDLE)
     {
      //--- imprimimos toda la información sobre el archivo
      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");
      //--- cerramos el archivo
      FileClose(handle);
     }
   else
      PrintFormat("%s file is not opened, ErrorCode = %d",InpFileName,GetLastError());
  }
//+------------------------------------------------------------------+
//| Visualización del valor de la propiedad del archivo
//+------------------------------------------------------------------+
void FileInfo(const int handle,const ENUM_FILE_PROPERTY_INTEGER id,
              long l,const string type)
  {
//--- obtenemos el valor de la propiedad
   ResetLastError();
   if((l=FileGetInteger(handle,id))!=-1)
     {
      //--- valor recibido, vamos a mostrarlo en el formato correcto
      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());
  }
```

Véase también

Operaciones con archivos, Propiedades de archivos

FileIsEnding

Determina el final de un archivo en el proceso de lectura.

```
bool  FileIsEnding(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

La función devuelve true, si se llega el final del archivo durante la lectura o en el proceso de mover el puntero de archivos.

Nota

Para detectar el fin del archivo, la función intenta leer la siguiente cadena del archivo. Si no la hay, la función devuelve true, de lo contrario - false.

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros de entrada
input string InpFileName="file.txt";     // nombre del archivo
input string InpDirectoryName="Data";    // nombre de la carpeta
input int    InpEncodingType=FILE_ANSI; // ANSI=32 o UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- imprimimos la ruta de la carpeta en la que vamos a trabajar
   PrintFormat("Trabajamos en la carpeta %s\\Files\\",TerminalInfoString(TERMINAL_DAT
//--- reseteamos el valor del error
   ResetLastError();
//--- abrimos el archivo de lectura (si el archivo no existe, se produce un error)
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpE
   if(file_handle!=INVALID_HANDLE)
     {
      //--- imprimimos el contenido del archivo
      while(!FileIsEnding(file_handle))
         Print(FileReadString(file_handle));
      //--- cerramos el archivo
      FileClose(file_handle);
     }
   else
      PrintFormat("Error, código = %d",GetLastError());
  }
```

FileIsLineEnding

Determina el fin de una línea en un archivo de texto en el proceso de lectura.

```
bool  FileIsLineEnding(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

Devuelve true, si se llega al final de la línea (símbolos CR-LF) durante la lectura de un archivo txt o csv.

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWriteString)

```
//+------------------------------------------------------------------+
//|                                        Demo_FileIsLineEnding.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Overbought & Oversold"
#property indicator_type1   DRAW_COLOR_BARS
#property indicator_color1  clrRed, clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parámetros para la lectura de datos
input string InpFileName="RSI.csv";   // nombre del archivo
input string InpDirectoryName="Data"; // nombre de la carpeta
//--- buferes de indicadores
double   open_buff[];
double   high_buff[];
double   low_buff[];
double   close_buff[];
double   color_buff[];
//--- variables de sobrecompra
int      ovb_ind=0;
int      ovb_size=0;
datetime ovb_time[];
//--- variables de sobreventa
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- variables de tamaños de arrays por defecto
   int ovb_def_size=100;
   int ovs_def_size=100;
//--- adjudicamos memoria para arrays
   ArrayResize(ovb_time,ovb_def_size);
   ArrayResize(ovs_time,ovs_def_size);
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("Archivo %s abierto para la lectura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      double value;
      //--- leemos los datos desde el archivo
      while(!FileIsEnding(file_handle))
        {
         //--- leemos el primer valor en la cadena
         value=FileReadNumber(file_handle);
         //--- leemos a diferentes arrays dependiendo del resultado de la función
         if(value>=70)
            ReadData(file_handle,ovb_time,ovb_size,ovb_def_size);
```

```
         else
            ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos leídos, archivo %s cerrado",InpFileName);
     }
   else
     {
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
      return(INIT_FAILED);
     }
//--- enlace de arrays
   SetIndexBuffer(0,open_buff,INDICATOR_DATA);
   SetIndexBuffer(1,high_buff,INDICATOR_DATA);
   SetIndexBuffer(2,low_buff,INDICATOR_DATA);
   SetIndexBuffer(3,close_buff,INDICATOR_DATA);
   SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);
//---- establecimiento de valores del indicador que no van a ser visibles en el gráfi
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Lectura de datos de la cadena de caracteres del archivo
//+------------------------------------------------------------------+
void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
  {
   bool flag=false;
//--- leemos hasta alcanzar el fin de la cadena o archivo
   while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
     {
      //--- desplazamos el carro al leer el número
      if(flag)
         FileReadNumber(file_handle);
      //--- recordamos la fecha actual
      arr[size]=FileReadDatetime(file_handle);
      size++;
      //--- si hace falta aumentamos el tamaño del array
      if(size==def_size)
        {
         def_size+=100;
         ArrayResize(arr,def_size);
        }
      //--- pasamos de la primera iteración
      flag=true;
     }
  }
//+------------------------------------------------------------------+
//| 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(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
//--- ciclo para las barras todavía no procesadas
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- por defecto 0
      open_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      close_buff[i]=0;
      color_buff[i]=0;
      //--- prueba de que si hay más datos
      if(ovb_ind<ovb_size)
         for(int j=ovb_ind;j<ovb_size;j++)
           {
            //--- si las fechas coinciden, la barra se encuentra en la zona de sobrec
            if(time[i]==ovb_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 0 - color rojo
               color_buff[i]=0;
               //--- aumentamos el contador
               ovb_ind=j+1;
               break;
              }
           }
      //--- prueba de que si hay más datos
      if(ovs_ind<ovs_size)
         for(int j=ovs_ind;j<ovs_size;j++)
           {
            //--- si las fechas coinciden, la barra se encuentra en la zona de sobrev
            if(time[i]==ovs_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 1 - color azul
               color_buff[i]=1;
               //--- aumentamos el contador
               ovs_ind=j+1;
               break;
              }
           }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Manejador del evento ChartEvent                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
```

```
                  )
  {
//--- variamos el grosor del indicador en función de la escala
   if(ChartGetInteger(0,CHART_SCALE)>3)
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,2);
   else
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
  }
```

Véase también

FileWriteString

FileReadArray

Lee los arrays de cualquier tipo, salvo los arrays literales (string) (puede ser un array de estructuras que no contienen las cadenas ni arrays dinámicos) de un archivo binario desde la posición actual del puntero de archivos.

```
uint  FileReadArray(
   int     file_handle,              // manejador del archivo
   void&   array[],                  // array para grabar
   int     start=0,                  // posición de inicio del array para grabar
   int     count=WHOLE_ARRAY         // cantidad para leer
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

array[]

[out] Array donde van a cargarse los datos.

start=0

[in] Posición de inicio para la escritura en el array.

count=WHOLE_ARRAY

[in] Número de elementos para la lectura.

Valor devuelto

Número de elementos leídos. Por defecto lee el array entero (count=WHOLE_ARRAY).

Nota

Un array de cadenas puede ser leído únicamente desde un archivo del tipo TXT. En caso de necesidad la función intenta aumentar el tamaño del array.

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWriteArray)

```
//--- mostramos la ventana de parámetros de entrada al arrancar el script
#property script_show_inputs
//--- parámetros de entrada
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| Estructura para guardar los datos de precios                     |
//+------------------------------------------------------------------+
struct prices
  {
   datetime          date; // fecha
   double            bid;  // precio bid
   double            ask;  // precio ask
  };
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- array de estructura
   prices arr[];
//--- ruta del archivo
   string path=InpDirectoryName+"//"+InpFileName;
//--- abrimos archivo
   ResetLastError();
   int file_handle=FileOpen(path,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      //--- leemos todos los datos desde el archivo al array
      FileReadArray(file_handle,arr);
      //--- obtenemos el tamaño del array
      int size=ArraySize(arr);
      //--- imprimimos los datos desde el array
      for(int i=0;i<size;i++)
         Print("Date = ",arr[i].date," Bid = ",arr[i].bid," Ask = ",arr[i].ask);
      Print("Total data = ",size);
      //--- cerramos el archivo
      FileClose(file_handle);
     }
   else
      Print("File open failed, error ",GetLastError());
  }
```

Véase también

Variables, FileWriteArray

FileReadBool

Lee de un archivo del tipo CSV una cadena desde la posición actual hasta el separador (o hasta el final de la cadena de caracteres) y convierte la cadena leída al valor del tipo bool.

```
bool  FileReadBool(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

La cadena leída puede tener valores "true", "false" o una representación simbólica de números enteros "0" o "1". El valor no nulo se convierte al lógico true. La función devuelve el valor convertido que se ha obtenido.

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWrite)

```
//+------------------------------------------------------------------+
//|                                              Demo_FileReadBool.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   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
//--- parámetros para la lectura de datos
input string InpFileName="MACD.csv";  // nombre del archivo
input string InpDirectoryName="Data"; // nombre de la carpeta
//--- variables globales
int      ind=0;        // índice
double   upbuff[];     // búfer de indicadores de flechas arriba
double   downbuff[];   // búfer de indicadores de flechas abajo
bool     sign_buff[]; // array de señales (true - compra, false - venta)
datetime time_buff[]; // array de la hora de accionamiento de señales
int      size=0;       // tamaño de arrays de señales
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("Archivo %s abierto para la lectura",InpFileName);
      //--- primero leemos el número de señales
      size=(int)FileReadNumber(file_handle);
      //--- adjudicamos la memoria para los arrays
      ArrayResize(sign_buff,size);
      ArrayResize(time_buff,size);
      //--- leemos los datos desde el archivo
      for(int i=0;i<size;i++)
        {
         //--- tiempo de la señal
         time_buff[i]=FileReadDatetime(file_handle);
         //--- valor de la señal
         sign_buff[i]=FileReadBool(file_handle);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
     }
   else
```

```
      {
       PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
       return(INIT_FAILED);
      }
//--- enlace de los arrays
   SetIndexBuffer(0,upbuff,INDICATOR_DATA);
   SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- establecemos el código del símbolo para dibujar en PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,241);
   PlotIndexSetInteger(1,PLOT_ARROW,242);
//---- establecimiento de valores del indicador que no van a ser visibles en el gráfi
   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(high,false);
   ArraySetAsSeries(low,false);
//--- ciclo para las barras todavía no procesadas
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- por defecto 0
      upbuff[i]=0;
      downbuff[i]=0;
      //--- prueba de que si hay más datos
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- si las fechas coinciden, utilizamos el valor desde el archivo
            if(time[i]==time_buff[j])
              {
               //--- dibujamos la flecha dependiendo de la señal
               if(sign_buff[j])
                  upbuff[i]=high[i];
               else
                  downbuff[i]=low[i];
               //--- aumentamos el contador
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Véase también

Tipo bool, FileWrite

FileReadDatetime

Lee de un archivo del tipo CSV una cadena de uno de los formatos: "YYYY.MM.DD HH:MI:SS", "YYYY.MM.DD" o "HH:MI:SS" - y la convierte al valor del tipo datetime.

```
datetime  FileReadDatetime(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

Valor del tipo datetime.

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWrite)

```
//+------------------------------------------------------------------+
//|                                        Demo_FileReadDateTime.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   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
//--- parámetros para la lectura de datos
input string InpFileName="MACD.csv";  // nombre del archivo
input string InpDirectoryName="Data"; // nombre de la carpeta
//--- variables globales
int      ind=0;        // índice
double   upbuff[];     // búfer de indicadores de flechas arriba
double   downbuff[];   // búfer de indicadores de flechas abajo
bool     sign_buff[];  // array de señales (true - compra, false - venta)
datetime time_buff[];  // array de la hora de accionamiento de señales
int      size=0;       // tamaño de arrays de señales
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("Archivo %s abierto para la lectura",InpFileName);
      //--- primero leemos el número de señales
      size=(int)FileReadNumber(file_handle);
      //--- adjudicamos la memoria para los arrays
      ArrayResize(sign_buff,size);
      ArrayResize(time_buff,size);
      //--- leemos los datos desde el archivo
      for(int i=0;i<size;i++)
        {
         //--- tiempo de la señal
         time_buff[i]=FileReadDatetime(file_handle);
         //--- valor de la señal
         sign_buff[i]=FileReadBool(file_handle);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
     }
   else
```

```
      {
       PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
       return(INIT_FAILED);
      }
//--- enlace de los arrays
   SetIndexBuffer(0,upbuff,INDICATOR_DATA);
   SetIndexBuffer(1,downbuff,INDICATOR_DATA);
//--- establecemos el código del símbolo para dibujar en PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,241);
   PlotIndexSetInteger(1,PLOT_ARROW,242);
//---- establecimiento de valores del indicador que no van a ser visibles en el gráfi
   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(high,false);
   ArraySetAsSeries(low,false);
//--- ciclo para las barras todavía no procesadas
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- por defecto 0
      upbuff[i]=0;
      downbuff[i]=0;
      //--- prueba de que si hay más datos
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- si las fechas coinciden, utilizamos el valor desde el archivo
            if(time[i]==time_buff[j])
              {
               //--- dibujamos la flecha dependiendo de la señal
               if(sign_buff[j])
                  upbuff[i]=high[i];
               else
                  downbuff[i]=low[i];
               //--- aumentamos el contador
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Véase también

Tipo datetime, StringToTime, TimeToString, FileWrite

FileReadDouble

Lee un número de doble precisión con punto flotante (double) de un archivo binario desde la posición actual del puntero de archivos.

```
double  FileReadDouble(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

Valor del tipo double.

Nota

Para obtener la información sobre el error hay que llamar a la función GetLastError().

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWriteDouble)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadDouble.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
#property indicator_separate_window
//--- parámetros para la lectura de datos
input string InpFileName="MA.csv";     // nombre del archivo
input string InpDirectoryName="Data"; // nombre de la carpeta
//--- variables globales
int      ind=0;
int      size=0;
double   ma_buff[];
datetime time_buff[];
//--- indicator buffer
double   buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("Archivo %s abierto para la lectura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- primero leemos cuántos datos contiene el archivo en total
      size=(int)FileReadDouble(file_handle);
      //--- adjudicamos la memoria para los arrays
      ArrayResize(ma_buff,size);
      ArrayResize(time_buff,size);
      //--- leemos los datos desde el archivo
      for(int i=0;i<size;i++)
        {
         time_buff[i]=(datetime)FileReadDouble(file_handle);
         ma_buff[i]=FileReadDouble(file_handle);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos leídos, archivo %s cerrado",InpFileName);
     }
   else
     {
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
      return(INIT_FAILED);
     }
//--- enlace del array al búfer de indicadores con el índice 0
```

```
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- establecimiento de valores del indicador que no van a mostrarse en el gráfico
   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);
//--- ciclo para las barras no procesadas todavía
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- por defecto 0
      buff[i]=0;
      //--- prueba de que si hay más datos
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- si las fechas coinciden, utilizamos el valor desde el archivo
            if(time[i]==time_buff[j])
              {
               buff[i]=ma_buff[j];
               //--- aumentamos el contador
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Véase también

Tipos reales (double, float), StringToDouble, DoubleToString, FileWriteDouble

FileReadFloat

Lee el número de precisión simple en punto flotante (float) de un archivo binario desde la posición actual.

```
float  FileReadFloat(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la funciónFileOpen().

Valor devuelto

Valor del tipo float.

Nota

Para obtener la información sobre el error hay que llamar a la función GetLastError().

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWriteFloat)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadFloat.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "CloseLine"
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  clrRed,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parámetros para la lectura de datos
input string InpFileName="Close.bin"; // nombre del archivo
input string InpDirectoryName="Data"; // nombre de la carpeta
//--- variables globales
int      ind=0;
int      size=0;
double   close_buff[];
datetime time_buff[];
//--- indicator buffers
double   buff[];
double   color_buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
   int def_size=100;
//--- adjudicamos la memoria para los arrays
   ArrayResize(close_buff,def_size);
   ArrayResize(time_buff,def_size);
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("Archivo %s abierto para la lectura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- leemos los datos desde el archivo
      while(!FileIsEnding(file_handle))
        {
         //--- leemos los valores de la hora y el precio
         time_buff[size]=(datetime)FileReadDouble(file_handle);
         close_buff[size]=(double)FileReadFloat(file_handle);
         size++;
         //--- aumentamos el tamaño de los arrays si están sobrecargados
         if(size==def_size)
           {
            def_size+=100;
            ArrayResize(close_buff,def_size);
            ArrayResize(time_buff,def_size);
           }
        }
      //--- cerramos el archivo
      FileClose(file_handle);
```

```
      PrintFormat("Datos leídos, archivo %s cerrado",InpFileName);
     }
   else
     {
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
      return(INIT_FAILED);
     }
//--- enlace de los arrays a los búferes de indicadores
   SetIndexBuffer(0,buff,INDICATOR_DATA);
   SetIndexBuffer(1,color_buff,INDICATOR_COLOR_INDEX);
//---- establecimiento de valores del indicador que no van a mostrarse en el gráfico
   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);
//--- ciclo para las barras no procesadas todavía
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- por defecto 0
      buff[i]=0;
      color_buff[i]=0; // el color rojo por defecto
      //--- prueba de que si hay más datos
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- si las fechas coinciden, utilizamos el valor desde el archivo
            if(time[i]==time_buff[j])
              {
               //--- obtenemos el precio
               buff[i]=close_buff[j];
               //--- si el precio actual supera el anterior, el color es azul
               if(buff[i-1]>buff[i])
                  color_buff[i]=1;
               //--- aumentamos el contador
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Véase también

Tipos reales (double, float), FileReadDouble, FileWriteFloat

FileReadInteger

La función lee de un archivo binario el valor del tipo int, short o char dependiendo de la longitud indicada en bytes. La lectura se realiza desde la posición actual del puntero de archivos.

```
int  FileReadInteger(
   int  file_handle,         // manejador del archivo
   int  size=INT_VALUE       // tamaño del tipo entero en bytes
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

size=INT_VALUE

[in] Número de bytes (hasta 4 inclusive) que hay que leer. Están previstas las siguientes constantes: CHAR_VALUE=1, SHORT_VALUE=2 y INT_VALUE=4, así la función puede leer el valor entero del tipo char, short o int.

Valor devuelto

Valor del tipo int. Es necesario convertir explicitamente el resultado de esta función a un tipo fuente, es decir a aquel tipo de datos que hace falta leer. Puesto que se devuelve el valor del tipo int, se puede convertirlo tranquilamente a cualquier valor entero. El puntero de archivo se desplaza a la cantidad de bytes leídos.

Nota

Cuando se lee menos de 4 bytes, el resultado obtenido será siempre positivo. Si se lee uno o dos bytes, se puede determinar exactamente el signo del número mediante la conversión explícita al tipo char (1 byte) o al tipo short (2 bytes), respectivamente. La obtención del signo para un número de tres bytes no es trivial, ya que no hay tipo base correspondiente.

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWriteInteger)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadInteger.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Trends"
#property indicator_type1   DRAW_SECTION
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- parámetros para la lectura de datos
input string InpFileName="Trend.bin"; // nombre del archivo
input string InpDirectoryName="Data"; // nombre de la carpeta
//--- variables globales
int      ind=0;
int      size=0;
datetime time_buff[];
//--- indicator buffers
double   buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
   int def_size=100;
//--- adjudicamos la memoria para el array
   ArrayResize(time_buff,def_size);
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("Archivo %s abierto para la lectura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- variables auxiliares
      int    arr_size;
      uchar  arr[];
      //--- leemos los datos desde el archivo
      while(!FileIsEnding(file_handle))
        {
         //--- averiguamos cuántos símbolos han sido utilizados para la escritura del
         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);
         //--- recordamos el valor del tiempo
         time_buff[size]=StringToTime(CharArrayToString(arr));
         size++;
         //--- aumentamos el tamaño de los arrays si están sobrecargados
         if(size==def_size)
           {
            def_size+=100;
            ArrayResize(time_buff,def_size);
           }
```

```
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos leídos, archivo %s cerrado",InpFileName);
     }
   else
     {
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
      return(INIT_FAILED);
     }
//--- enlace del array al búfer de indicadores
   SetIndexBuffer(0,buff,INDICATOR_DATA);
//---- establecimiento de valores del indicador que no van a mostrarse en el gráfico
   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);
   ArraySetAsSeries(close,false);
//--- ciclo para las barras no procesadas todavía
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- por defecto 0
      buff[i]=0;
      //--- prueba de que si hay más datos
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- si las fechas coinciden, utilizamos el valor desde el archivo
            if(time[i]==time_buff[j])
              {
               //--- obtenemos el precio
               buff[i]=close[i];
               //--- aumentamos el contador
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Véase también

IntegerToString, StringToInteger, Tipos enteros, FileWriteInteger

FileReadLong

Lee el número entero del tipo long (8 bytes) desde la posición actual del puntero de archivos de un archivo binario.

```
long  FileReadLong(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

Valor del tipo long.

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWriteLong)

```
//+------------------------------------------------------------------+
//|                                            Demo_FileReadLong.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_buffers 1
#property indicator_plots   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
//--- parámetros para la lectura de datos
input string InpFileName="Volume.bin"; // nombre del archivo
input string InpDirectoryName="Data";  // nombre de la carpeta
//--- variables globales
int      ind=0;
int      size=0;
long     volume_buff[];
datetime time_buff[];
//--- indicator buffers
double   buff[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN);
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("Archivo %s abierto para la escritura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- primero leemos cuántos datos contiene el archivo en total
      size=(int)FileReadLong(file_handle);
      //--- adjudicamos la memoria para los arrays
      ArrayResize(volume_buff,size);
      ArrayResize(time_buff,size);
      //--- leemos los datos desde el archivo
      for(int i=0;i<size;i++)
        {
         time_buff[i]=(datetime)FileReadLong(file_handle);
         volume_buff[i]=FileReadLong(file_handle);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos leídos, archivo %s cerrado",InpFileName);
     }
   else
     {
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
      return(INIT_FAILED);
     }
//--- enlace del array al búfer de indicadores con el índice 0
   SetIndexBuffer(0,buff,INDICATOR_DATA);
```

```
//---- establecimiento de valores del indicador que no van a mostrarse en el gráfico
   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);
//--- ciclo para las barras no procesadas todavía
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- por defecto 0
      buff[i]=0;
      //--- prueba de que si hay más datos
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- si las fechas coinciden, utilizamos el valor desde el archivo
            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);
  }
```

Véase también

Tipos enteros, FileReadInteger, FileWriteLong

FileReadNumber

Lee de un archivo del tipo CSV una cadena desde la posición actual hasta el separador (o hasta el final de la línea de texto) y convierte la cadena leída al valor del tipo double.

```
double  FileReadNumber(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

Valor del tipo double.

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWriteString)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadNumber.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Overbought & Oversold"
#property indicator_type1   DRAW_COLOR_BARS
#property indicator_color1  clrRed, clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  2
//--- parámetros para la lectura de datos
input string InpFileName="RSI.csv";   // nombre del archivo
input string InpDirectoryName="Data"; // nombre de la carpeta
//--- buferes de indicadores
double   open_buff[];
double   high_buff[];
double   low_buff[];
double   close_buff[];
double   color_buff[];
//--- variables de sobrecompra
int      ovb_ind=0;
int      ovb_size=0;
datetime ovb_time[];
//--- variables de sobreventa
int      ovs_ind=0;
int      ovs_size=0;
datetime ovs_time[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- variables de tamaños de arrays por defecto
   int ovb_def_size=100;
   int ovs_def_size=100;
//--- adjudicamos memoria para arrays
   ArrayResize(ovb_time,ovb_def_size);
   ArrayResize(ovs_time,ovs_def_size);
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_CSV|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("Archivo %s abierto para la lectura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      double value;
      //--- leemos los datos desde el archivo
      while(!FileIsEnding(file_handle))
        {
         //--- leemos el primer valor en la cadena
         value=FileReadNumber(file_handle);
         //--- leemos a diferentes arrays dependiendo del resultado de la función
         if(value>=70)
            ReadData(file_handle,ovb_time,ovb_size,ovb_def_size);
```

```
         else
            ReadData(file_handle,ovs_time,ovs_size,ovs_def_size);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos leídos, archivo %s cerrado",InpFileName);
     }
   else
     {
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
      return(INIT_FAILED);
     }
//--- enlace de arrays
   SetIndexBuffer(0,open_buff,INDICATOR_DATA);
   SetIndexBuffer(1,high_buff,INDICATOR_DATA);
   SetIndexBuffer(2,low_buff,INDICATOR_DATA);
   SetIndexBuffer(3,close_buff,INDICATOR_DATA);
   SetIndexBuffer(4,color_buff,INDICATOR_COLOR_INDEX);
//---- establecimiento de valores del indicador que no van a ser visibles en el gráfi
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Lectura de datos de la cadena de caracteres del archivo
//+------------------------------------------------------------------+
void ReadData(const int file_handle,datetime &arr[],int &size,int &def_size)
  {
   bool flag=false;
//--- leemos hasta alcanzar el fin de la cadena o archivo
   while(!FileIsLineEnding(file_handle) && !FileIsEnding(file_handle))
     {
      //--- desplazamos el carro al leer el número
      if(flag)
         FileReadNumber(file_handle);
      //--- recordamos la fecha actual
      arr[size]=FileReadDatetime(file_handle);
      size++;
      //--- si hace falta aumentamos el tamaño del array
      if(size==def_size)
        {
         def_size+=100;
         ArrayResize(arr,def_size);
        }
      //--- pasamos de la primera iteración
      flag=true;
     }
  }
//+------------------------------------------------------------------+
//| 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(open,false);
   ArraySetAsSeries(high,false);
   ArraySetAsSeries(low,false);
   ArraySetAsSeries(close,false);
//--- ciclo para las barras todavía no procesadas
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- por defecto 0
      open_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      close_buff[i]=0;
      color_buff[i]=0;
      //--- prueba de que si hay más datos
      if(ovb_ind<ovb_size)
         for(int j=ovb_ind;j<ovb_size;j++)
           {
            //--- si las fechas coinciden, la barra se encuentra en la zona de sobrec
            if(time[i]==ovb_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 0 - color rojo
               color_buff[i]=0;
               //--- aumentamos el contador
               ovb_ind=j+1;
               break;
              }
           }
      //--- prueba de que si hay más datos
      if(ovs_ind<ovs_size)
         for(int j=ovs_ind;j<ovs_size;j++)
           {
            //--- si las fechas coinciden, la barra se encuentra en la zona de sobrev
            if(time[i]==ovs_time[j])
              {
               open_buff[i]=open[i];
               high_buff[i]=high[i];
               low_buff[i]=low[i];
               close_buff[i]=close[i];
               //--- 1 - color azul
               color_buff[i]=1;
               //--- aumentamos el contador
               ovs_ind=j+1;
               break;
              }
           }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Manejador del evento ChartEvent                                  |
//+------------------------------------------------------------------+
void OnChartEvent(const int id,
                  const long &lparam,
                  const double &dparam,
                  const string &sparam
```

```
                  )
  {
//--- variamos el grosor del indicador en función de la escala
   if(ChartGetInteger(0,CHART_SCALE)>3)
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,2);
   else
      PlotIndexSetInteger(0,PLOT_LINE_WIDTH,1);
  }
```

Véase también

FileWriteString

FileReadString

Lee del archivo una cadena desde la posición actual del puntero de archivos.

```
string  FileReadString(
   int  file_handle,     // manejador del archivo
   int  length=-1        // longitud de la cadena
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

length=-1

[in] Número de caracteres para leer.

Valor devuelto

Cadena leída (string).

Nota

Cuando leemos de un archivo bin es obligatorio indicar la longitud de la cadena. Cuando leemos de un archivo txt no hace falta especificar la longitud de la cadena; la cadena será leída desde la posición actual hasta el salto de línea "\r\n". Cuando leemos de un archivo csv tampoco hace falta especificar la longitud de la cadena; la cadena será leída desde la posición actual hasta el separador más cercano o hasta el salto de línea.

Si el archivo está abierto con la bandera FILE_ANSI, la cadena leída se convierte a Unicode.

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWriteInteger)

```
///--- mostramos la ventana de los parámetros de entrada al arrancar el script
#property script_show_inputs
//--- parámetros para la lectura de datos
input string InpFileName="Trend.bin"; // nombre del archivo
input string InpDirectoryName="Data"; // nombre de la carpeta
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la lectura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- variables auxiliares
      int    str_size;
      string str;
      //--- leemos los datos desde el archivo
      while(!FileIsEnding(file_handle))
        {
         //--- averiguamos cuántos símbolos han sido utilizados para la escritura del
         str_size=FileReadInteger(file_handle,INT_VALUE);
         //--- leemos la cadena
         str=FileReadString(file_handle,str_size);
         //--- imprimimos la cadena
         PrintFormat(str);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos leídos, archivo %s cerrado",InpFileName);
     }
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
  }
```

Véase también

Tipo string, Conversión de datos, FileWriteInteger

FileReadStruct

Lee de un archivo binario el contenido en una estructura que ha sido pasada como un parámetro. La lectura se realiza desde la posición actual del puntero de archivos.

```
uint  FileReadStruct(
   int          file_handle,        // manejador del archivo
   const void&  struct_object,      // estructura de destino para la lectura
   int          size=-1             // tamaño de estructura en bytes
   );
```

Parámetros

file_handle

[in] Descriptor de archivos de un archivo binario abierto.

struct_object

[out] Objeto de esta estructura. La estructura no debe contener cadenas, arrays dinámicos y funciones virtuales.

size=-1

[in] Cantidad de bytes que hay que leer. Si el tamaño no se especifica o la cantidad de bytes especificada supera el tamaño de la estructura, entonces se usa el tamaño exacto de la estructura indicada.

Valor devuelto

En caso de éxito la función devuelve el número de bytes leído. El puntero de archivos se mueve a esta misma cantidad de bytes.

Ejemplo (se utiliza el archivo conseguido como resultado de trabajo del ejemplo para la función FileWriteStruct)

```
//+------------------------------------------------------------------+
//|                                          Demo_FileReadStruct.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   1
//---- plot Label1
#property indicator_label1  "Candles"
#property indicator_type1   DRAW_CANDLES
#property indicator_color1  clrOrange
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
#property indicator_separate_window
//--- parámetros para recibir datos
input string  InpFileName="EURUSD.txt"; // nombre del archivo
input string  InpDirectoryName="Data";  // nombre de la carpeta
//+------------------------------------------------------------------+
//| Estructura para almacenar los datos de la vela                   |
//+------------------------------------------------------------------+
struct candlesticks
  {
   double            open;  // precio de apertura
   double            close; // precio de cierre
   double            high;  // precio máximo
   double            low;   // precio mínimo
   datetime          date;  // fecha
  };
//--- búferes de indicadores
double        open_buff[];
double        close_buff[];
double        high_buff[];
double        low_buff[];
//--- variables globales
candlesticks cand_buff[];
int          size=0;
int          ind=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
   int default_size=100;
   ArrayResize(cand_buff,default_size);
//--- abrimos el archivo
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_BIN|FILE_
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("Archivo %s abierto para la lectura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_COMMOND
      //--- leemos los datos desde el archivo
      while(!FileIsEnding(file_handle))
        {
         //--- escribimos los datos en el array
         FileReadStruct(file_handle,cand_buff[size]);
         size++;
```

```
         //--- comprobamos si el array está sobrecargado
         if(size==default_size)
           {
            //--- aumentar la dimensión del array
            default_size+=100;
            ArrayResize(cand_buff,default_size);
           }
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos leídos, archivo %s cerrado",InpFileName);
     }
   else
     {
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
      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);
//--- valor vacío
   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);
//--- ciclo para las velas no procesadas todavía
   for(int i=prev_calculated;i<rates_total;i++)
     {
      //--- por defecto 0
      open_buff[i]=0;
      close_buff[i]=0;
      high_buff[i]=0;
      low_buff[i]=0;
      //--- prueba de que si hay más datos
      if(ind<size)
        {
         for(int j=ind;j<size;j++)
           {
            //--- si las fechas coinciden, utilizamos el valor desde el archivo
            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;
               //--- aumentamos el contador
               ind=j+1;
               break;
              }
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Véase también

Estructuras y clases, FileWriteStruct

FileSeek

Mueve la posición del puntero de archivos a una cantidad de bytes especificada respecto a la posición indicada.

```
void  FileSeek(
   int                  file_handle,     // manejador del archivo
   long                 offset,          // en bytes
   ENUM_FILE_POSITION   origin           // posición de referencia
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

offset

[in] Desplazamiento en bytes (puede también adquirir un valor negativo).

origin

[in] Punto de referencia para el desplazamiento. Puede ser uno de los valores de la enumeración ENUM_FILE_POSITION.

Valor devuelto

No hay valor devuelto.

Nota

Si el resultado de ejecución de la función FileSeek() es un desplazamiento negativo (sobrepasando "el límite izquierdo" del archivo), el puntero de archivos será puesto en el principio del archivo.

Si la posición se pone fuera del "límite derecho" del archivo (más que el tamaño del archivo), la siguiente escritura en el archivo será realizada no desde el final sino desde la posición puesta. En este caso entre el final anterior del archivo y posición puesta habrán unos valores indeterminados.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_FileSeek.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros de entrada
input string InpFileName="file.txt";     // nombre del archivo
input string InpDirectoryName="Data";    // nombre de la carpeta
input int    InpEncodingType=FILE_ANSI; // ANSI=32 o UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- especificamos el valor de la variable para generar los números aleatorios
   _RandomSeed=GetTickCount();
//--- variable para las posiciones de inicio de cadenas
   ulong pos[];
   int   size;
//--- reseteamos el valor del error
   ResetLastError();
//--- abrimos el archivo
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpE
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la lectura",InpFileName);
      //--- obtenemos la posición de inicio para cada cadena de caracteres en el arch
      GetStringPositions(file_handle,pos);
      //--- averiguamos el número total de cadenas en el archivo
      size=ArraySize(pos);
      if(!size)
        {
         //--- si en el archivo no hay cadenas, finalizamos el trabajo
         PrintFormat("¡El archivo %s esta vacío!",InpFileName);
         FileClose(file_handle);
         return;
        }
      //--- elegimos el número de una cadena al azar
      int ind=MathRand()%size;
      //--- desplazamos la posición al inicio de esta cadena
      FileSeek(file_handle,pos[ind],SEEK_SET);
      //--- leemos e imprimimos la cadena con el número ind
      PrintFormat("Texto de la cadena con el número %d: \"%s\"",ind,FileReadString(fi
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("El archivo %s ha sido cerrado",InpFileName);
     }
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
  }
//+------------------------------------------------------------------+
//| La función determina las posiciones de inicio para cada una de las cadenas en el
//| las coloca en el array arr                                       |
//+------------------------------------------------------------------+
void GetStringPositions(const int handle,ulong &arr[])
  {
```

```
//--- tamaño del array por defecto
   int def_size=127;
//--- adjudicamos la memoria para el array
   ArrayResize(arr,def_size);
//--- contador de cadenas
   int i=0;
//--- si no es el fin del archivo, entonces por lo menos hay una cadena
   if(!FileIsEnding(handle))
     {
      arr[i]=FileTell(handle);
      i++;
     }
   else
      return; // el archivo está vacío, salimos
//--- determinamos el desplazamiento en bytes dependiendo de la codificación
   int shift;
   if(FileGetInteger(handle,FILE_IS_ANSI))
      shift=1;
   else
      shift=2;
//--- repasamos las cadenas en el ciclo
   while(1)
     {
      //--- leemos la cadena
      FileReadString(handle);
      //--- prueba en el fin del archivo
      if(!FileIsEnding(handle))
        {
         //--- recordamos la posición de la siguiente cadena
         arr[i]=FileTell(handle)+shift;
         i++;
         //--- aumentamos el tamaño del array si está sobrecargado
         if(i==def_size)
           {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
           }
        }
      else
         break; // fin del archivo, salimos
     }
//--- definimos el tamaño real del array
   ArrayResize(arr,i);
  }
```

FileSize

Devuelve el tamaño del archivo en bytes.

```
ulong  FileSize(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

Valor del tipo int.

Nota

Para obtener la información sobre el error hay que llamar a la función GetLastError().

Ejemplo:

```
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros de entrada
input ulong  InpThresholdSize=20;        // umbral del tamaño de archivos en kilobyte
input string InpBigFolderName="big";     // carpeta para archivos grandes
input string InpSmallFolderName="small"; // carpeta para archivos pequeños
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string   file_name;      // variable para almacenar los nombres de archivos
   string   filter="*.csv"; // filtro para buscar archivos
   ulong    file_size=0;    // tamaño del archivo en bytes
   int      size=0;         // número de archivos
//--- imprimimos la ruta de la carpeta en la que vamos a trabajar
   PrintFormat("Trabajamos en la carpeta %s\\Files\\",TerminalInfoString(TERMINAL_COM
//--- recepción del manejador de búsqueda en la raíz de la carpeta común de todos los
   long search_handle=FileFindFirst(filter,file_name,FILE_COMMON);
//--- comprobamos si la función FileFindFirst()ha terminado su trabajo con éxito
   if(search_handle!=INVALID_HANDLE)
     {
      //--- movemos los archivos en el ciclo dependiendo de su tamaño
      do
        {
         //--- abrimos el archivo
         ResetLastError();
         int file_handle=FileOpen(file_name,FILE_READ|FILE_CSV|FILE_COMMON);
         if(file_handle!=INVALID_HANDLE)
           {
            //--- obtenemos el tamaño del archivo
            file_size=FileSize(file_handle);
            //--- cerramos el archivo
            FileClose(file_handle);
           }
         else
           {
            PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",file_na
            continue;
           }
         //--- imprimimos el tamaño del archivo
         PrintFormat("El tamaño del archivo %s es de %d bytes",file_name,file_size);
         //--- definimos la ruta para mover el archivo
         string path;
         if(file_size>InpThresholdSize*1024)
            path=InpBigFolderName+"//"+file_name;
         else
            path=InpSmallFolderName+"//"+file_name;
         //--- desplazamos el archivo
         ResetLastError();
         if(FileMove(file_name,FILE_COMMON,path,FILE_REWRITE|FILE_COMMON))
            PrintFormat("El archivo %s ha sido desplazado",file_name);
         else
            PrintFormat("Error, código = %d",GetLastError());
        }
      while(FileFindNext(search_handle,file_name));
      //--- cerramos el manejador de búsqueda
      FileFindClose(search_handle);
     }
   else
      Print("Files not found!");
```

```
}
```

FileTell

Devuelve la posición actual del puntero de archivos de un archivo abierto correspondiente.

```
ulong  FileTell(
   int  file_handle      // manejador del archivo
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

Valor devuelto

Posición actual del descriptor de archivos desde el principio del archivo.

Nota

Para obtener la información sobre el error hay que llamar a la función GetLastError().

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_FileTell.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros de entrada
input string InpFileName="file.txt";     // nombre del archivo
input string InpDirectoryName="Data";    // nombre de la carpeta
input int    InpEncodingType=FILE_ANSI; // ANSI=32 o UNICODE=64
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- especificamos el valor de la variable para generar los números aleatorios
   _RandomSeed=GetTickCount();
//--- variable para las posiciones de inicio de cadenas
   ulong pos[];
   int   size;
//--- reseteamos el valor del error
   ResetLastError();
//--- abrimos el archivo
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_TXT|InpE
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la lectura",InpFileName);
      //--- obtenemos la posición de inicio para cada cadena de caracteres en el arch
      GetStringPositions(file_handle,pos);
      //--- averiguamos el número total de cadenas en el archivo
      size=ArraySize(pos);
      if(!size)
        {
         //--- si en el archivo no hay cadenas, finalizamos el trabajo
         PrintFormat("¡El archivo %s esta vacío!",InpFileName);
         FileClose(file_handle);
         return;
        }
      //--- elegimos el número de una cadena al azar
      int ind=MathRand()%size;
      //--- desplazamos la posición al inicio de esta cadena
      FileSeek(file_handle,pos[ind],SEEK_SET);
      //--- leemos e imprimimos la cadena con el número ind
      PrintFormat("Texto de la cadena con el número %d: \"%s\"",ind,FileReadString(fi
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("El archivo %s ha sido cerrado",InpFileName);
     }
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
  }
//+------------------------------------------------------------------+
//| La función determina las posiciones de inicio para cada una de las cadenas en el
//| las coloca en el array arr                                       |
//+------------------------------------------------------------------+
void GetStringPositions(const int handle,ulong &arr[])
  {
```

```
//--- tamaño del array por defecto
   int def_size=127;
//--- adjudicamos la memoria para el array
   ArrayResize(arr,def_size);
//--- contador de cadenas
   int i=0;
//--- si no es el fin del archivo, entonces por lo menos hay una cadena
   if(!FileIsEnding(handle))
     {
      arr[i]=FileTell(handle);
      i++;
     }
   else
      return; // el archivo está vacío, salimos
//--- determinamos el desplazamiento en bytes dependiendo de la codificación
   int shift;
   if(FileGetInteger(handle,FILE_IS_ANSI))
      shift=1;
   else
      shift=2;
//--- repasamos las cadenas en el ciclo
   while(1)
     {
      //--- leemos la cadena
      FileReadString(handle);
      //--- prueba en el fin del archivo
      if(!FileIsEnding(handle))
        {
         //--- recordamos la posición de la siguiente cadena
         arr[i]=FileTell(handle)+shift;
         i++;
         //--- aumentamos el tamaño del array si está sobrecargado
         if(i==def_size)
           {
            def_size+=def_size+1;
            ArrayResize(arr,def_size);
           }
        }
      else
         break; // fin del archivo, salimos
     }
//--- definimos el tamaño real del array
   ArrayResize(arr,i);
  }
```

FileWrite

Esta función se utiliza para escribir los datos en un archivo del tipo CSV o TXT. Si el delimitador no es igual a 0, entonces se inserta entre los datos automáticamente. Después de la escritura en el archivo, se añade el salto de línea "\r\n".

```
uint  FileWrite(
   int  file_handle,     // manejador del archivo
   ...                   // lista de parámetros a escribir
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

...

[in] Lista de parámetros separados por coma para escribirlos en el archivo. El número de parámetros a introducir no puede superar 63.

Valor devuelto

Número de bytes escritos.

Nota

En la salida los datos numéricos se convierten al formato de texto (ver la función Print()). Los datos del tipo double se visualizan con la precisión de hasta 16 dígitos decimales después del punto, además, los datos pueden ser visualizados en el formato tradicional o científico, dependiendo de cuál de ellos va a ser más compacto. Los datos del tipo float se visualizan con 5 dígitos decimales después del punto. Para visualizar los números reales con otra precisión o en un formato explícitamente especificado hay que usar la función DoubleToString().

Los números del tipo bool se visualizan como cadenas "true" o "false". Los números del tipo datetime se visualizan en el formato "YYYY.MM.DD HH:MI:SS".

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_FileWrite.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros para recibir datos desde el terminal
input string             InpSymbolName="EURUSD";           // par de divisas
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;        // período de tiempo
input int                InpFastEMAPeriod=12;              // período de la EMA rápid
input int                InpSlowEMAPeriod=26;              // período de la EMA lenta
input int                InpSignalPeriod=9;                // período del promedio de
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;      // tipo del precio
input datetime           InpDateStart=D'2012.01.01 00:00'; // fecha de inicio del cop
//--- parámetros para la escritura de datos en el archivo
input string             InpFileName="MACD.csv";  // nombre del archivo
input string             InpDirectoryName="Data"; // nombre de la carpeta
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish; // fecha del fin de copiado de datos
   bool     sign_buff[]; // array de señales (true - compra, false - venta)
   datetime time_buff[]; // array del tiempo de llegada de señales
   int      sign_size=0; // tamaño de arrays de las señales
   double   macd_buff[]; // array de los valores del indicador
   datetime date_buff[]; // array de las fechas del indicador
   int      macd_size=0; // tamaño de arrays del indicador
//--- tiempo de finalización - actual
   date_finish=TimeCurrent();
//--- obtenemos el manejador del indicador MACD
   ResetLastError();
   int macd_handle=iMACD(InpSymbolName,InpSymbolPeriod,InpFastEMAPeriod,InpSlowEMAPer
   if(macd_handle==INVALID_HANDLE)
     {
      //--- fallo al obtener el manejador del indicador
      PrintFormat("Error a la hora de obtener el manejador del indicador. Código del
      return;
     }
//--- seguimos en el ciclo hasta que el indicador no termine de calcular todos sus va
   while(BarsCalculated(macd_handle)==-1)
      Sleep(10); // retardo para que al indicador le de tiempo a calcular sus valores
//--- copiamos los valores del indicador durante un período determinado
   ResetLastError();
   if(CopyBuffer(macd_handle,0,InpDateStart,date_finish,macd_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores del indicador. Código del error = %d",
      return;
     }
//--- copiamos el tiempo correspondiente para los valores del indicador
   ResetLastError();
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,date_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores del tiempo. Código del error = %d",Get
      return;
     }
```

```
//--- liberamos la memoria que ocupa el indicador
   IndicatorRelease(macd_handle);
//--- obtenemos el tamaño del búfer
   macd_size=ArraySize(macd_buff);
//--- vamos a analizar los datos y guardar las señales del indicador en los arrays
   ArrayResize(sign_buff,macd_size-1);
   ArrayResize(time_buff,macd_size-1);
   for(int i=1;i<macd_size;i++)
     {
      //--- señales de compra
      if(macd_buff[i-1]<0 && macd_buff[i]>=0)
        {
         sign_buff[sign_size]=true;
         time_buff[sign_size]=date_buff[i];
         sign_size++;
        }
      //--- señales de venta
      if(macd_buff[i-1]>0 && macd_buff[i]<=0)
        {
         sign_buff[sign_size]=false;
         time_buff[sign_size]=date_buff[i];
         sign_size++;
        }
     }
//--- abrimos el archivo para escribir los valores del indicador (si no existe, se cr
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la escritura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- primero apuntamos el número de señales
      FileWrite(file_handle,sign_size);
      //--- apuntamos en el archivo el tiempo de señales y sus valores
      for(int i=0;i<sign_size;i++)
         FileWrite(file_handle,time_buff[i],sign_buff[i]);
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos grabados, el archivo %s cerrado",InpFileName);
     }
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
  }
```

Véase también

Comment, Print, StringFormat

FileWriteArray

Esta función escribe los arrays de cualquier tipo, excepto los arrays para las cadenas, en un archivo del tipo BIN (puede ser un array de estructuras que no contienen cadenas ni arrays dinámicos).

```
uint  FileWriteArray(
   int           file_handle,          // manejador del archivo
   const void&   array[],              // matriz
   int           start=0,              // índice inicial en el array
   int           count=WHOLE_ARRAY     // número de elementos
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

array[]

[out] Array a grabar.

start=0

[in] Índice inicial en el array (número del primer elemento a grabar).

count=WHOLE_ARRAY

[in] Número de elementos que se graban (WHOLE_ARRAY significa que se graban todos los elementos empezando desde el número start hasta el final del array).

Valor devuelto

Número de elementos grabados.

Nota

Un array de cadenas puede ser grabado sólo en un archivo del tipo TXT. En este caso las cadenas automáticamente se acaban con los caracteres de salto de línea "\r\n". Dependiendo del tipo de archivo ANSI o UNICODE, las cadenas se convierten a la codificación ansi o no.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteArray.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- parámetros de entrada
input string InpFileName="data.bin";
input string InpDirectoryName="SomeFolder";
//+------------------------------------------------------------------+
//| Estructura para almacenar los datos sobre los precios
//+------------------------------------------------------------------+
struct prices
  {
   datetime          date; // fecha
   double            bid;  // precio bid
   double            ask;  // precio ask
  };
//--- variables globales
int    count=0;
int    size=20;
string path=InpDirectoryName+"//"+InpFileName;
prices arr[];
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- adjudicación de la memoria para el array
   ArrayResize(arr,size);
//---
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- escribir count cadenas restantes, si count<n
   WriteData(count);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
void OnTick()
  {
//--- guardamos los datos en el array
   arr[count].date=TimeCurrent();
   arr[count].bid=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   arr[count].ask=SymbolInfoDouble(Symbol(),SYMBOL_ASK);
//--- mostramos los datos actuales
   Print("Date = ",arr[count].date," Bid = ",arr[count].bid," Ask = ",arr[count].ask)
//--- aumentamos el contador
   count++;
//--- si el array ya esta lleno, escribimos los datos en el archivo y lo ponemos a ce
   if(count==size)
     {
      WriteData(size);
      count=0;
```

```
      }
   }
//+------------------------------------------------------------------+
//| Escritura de n elementos del array en el archivo                 |
//+------------------------------------------------------------------+
void WriteData(const int n)
   {
//--- abrimos el archivo
    ResetLastError();
    int handle=FileOpen(path,FILE_READ|FILE_WRITE|FILE_BIN);
    if(handle!=INVALID_HANDLE)
      {
       //--- escribimos los datos del array al final del archivo
       FileSeek(handle,0,SEEK_END);
       FileWriteArray(handle,arr,0,n);
       //--- cerramos el archivo
       FileClose(handle);
      }
    else
       Print("Failed to open the file, error ",GetLastError());
   }
```

Véase también

Variables, FileSeek

FileWriteDouble

Escribe el valor del parámetro del tipo double desde la posición actual del puntero de archivos en un archivo.

```
uint  FileWriteDouble(
   int     file_handle,     // manejador del archivo
   double  value            // valor para escribir
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

value

[in] Valor del tipo double.

Valor devuelto

En caso del éxito la función devuelve el número de bytes escritos (en este caso sizeof(double)=8). El puntero de archivos se mueve a esta misma cantidad de bytes.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                        Demo_FileWriteDouble.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros para obtener datos desde el terminal
input string             InpSymbolName="EURJPY";           // par de divisas
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_M15;       // período de tiempo
input int                InpMAPeriod=10;                   // período de suavizado
input int                InpMAShift=0;                     // desplazamiento del indi
input ENUM_MA_METHOD     InpMAMethod=MODE_SMA;             // tipo de suavizado
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;      // tipo del precio
input datetime           InpDateStart=D'2013.01.01 00:00'; // fecha de inicio del cop
//--- parámetros para la escritura de datos en el archivo
input string             InpFileName="MA.csv";    // nombre del archivo
input string             InpDirectoryName="Data"; // nombre de la carpeta
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   ma_buff[];
   datetime time_buff[];
   int      size;
//--- obtenemos el indicador del manejador MA
   ResetLastError();
   int ma_handle=iMA(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpMAShift,InpMAMethod
   if(ma_handle==INVALID_HANDLE)
     {
      //--- fallo al obtener el manejador del indicador
      PrintFormat("Fallo al obtener el manejador del indicador. Código del error = %d
      return;
     }
//--- seguimos en el ciclo hasta que el indicador no termine de calcular todos sus va
   while(BarsCalculated(ma_handle)==-1)
      Sleep(20); // retardo para que al indicador le de tiempo a calcular sus valores
   PrintFormat("En el archivo serán escritos los valores del indicador a partir del %
//--- copiamos los valpres del indicador
   ResetLastError();
   if(CopyBuffer(ma_handle,0,InpDateStart,date_finish,ma_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores del indicador. Código del error = %d",
      return;
     }
//--- copiamos el tiempo de aparición de barras correspondientes
   ResetLastError();
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores del tiempo. Código del error = %d",Get
      return;
     }
//--- obtenemos el tamaño del búfer
   size=ArraySize(ma_buff);
//--- liberamos la memoria que ocupa el indicador
   IndicatorRelease(ma_handle);
```

```
//--- abrimos el archivo para escribir los valores del indicador (si no existe, se cr
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la escritura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- primero escribimos el tamaño de la muestra de datos
      FileWriteDouble(file_handle,(double)size);
      //--- escribimos en el archivo el tiempo y los valores del indicador
      for(int i=0;i<size;i++)
        {
         FileWriteDouble(file_handle,(double)time_buff[i]);
         FileWriteDouble(file_handle,ma_buff[i]);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos grabados, el archivo %s cerrado",InpFileName);
     }
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
  }
```

Véase también

Tipos reales (double, float)

FileWriteFloat

Escribe el valor del parámetro del tipo float desde la posición actual del puntero de archivos en un archivo binario.

```
uint  FileWriteFloat(
   int    file_handle,     // manejador del archivo
   float  value            // valor para escribir
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

value

[in] Valor del tipo float.

Valor devuelto

En caso del éxito la función devuelve el número de bytes escritos (en este caso sizeof(float)=4). El puntero de archivos se mueve a esta misma cantidad de bytes.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteFloat.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros para obtener datos desde el terminal
input string          InpSymbolName="EURUSD";           // par de divisas
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_M15;       // período de tiempo
input datetime        InpDateStart=D'2013.01.01 00:00'; // fecha de inicio del copiad
//--- parámetros para la escritura de datos en el archivo
input string          InpFileName="Close.bin"; // nombre del archivo
input string          InpDirectoryName="Data"; // nombre de la carpeta
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   close_buff[];
   datetime time_buff[];
   int      size;
//--- reseteamos el valor del error
   ResetLastError();
//--- copiamos el precio de cierre para cada barra
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-
     {
      PrintFormat("Fallo al copiar los valores de los precios de cierre. Código del e
      return;
     }
//--- copiamos el tiempo para cada barra
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Fallo al copiar el valor del tiempo. Código del error = %d",GetLas
      return;
     }
//--- obtenemos el tamaño del búfer
   size=ArraySize(close_buff);
//--- abrimos el archivo para escribir los valores (si no existe, se crea automáticam
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la escritura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- escribimos el tiempo y los valores de precios de cierre en el archivo
      for(int i=0;i<size;i++)
        {
         FileWriteDouble(file_handle,(double)time_buff[i]);
         FileWriteFloat(file_handle,(float)close_buff[i]);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos grabados, el archivo %s cerrado",InpFileName);
     }
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
```

```
}
```

Véase también

Tipos reales (double, float), FileWriteDouble

FileWriteInteger

Escribe el valor del parámetro del tipo int desde la posición actual del puntero de archivos en un archivo binario.

```
uint  FileWriteInteger(
   int  file_handle,         // manejador del archivo
   int  value,               // valor para escribir
   int  size=INT_VALUE       // tamaño en bytes
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

value

[in] Valor entero.

size=INT_VALUE

[in] Cantidad de bytes (hasta 4 inclusive) que hay que escribir. Están previstas las constantes siguientes: CHAR_VALUE=1, SHORT_VALUE=2 y INT_VALUE=4, de esta manera, la función puede escribir el valor entero del tipo char, uchar, short, ushort, int o uint.

Valor devuelto

En caso del éxito la función devuelve el número de bytes escritos. El puntero de archivos se mueve a esta misma cantidad de bytes.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                      Demo_FileWriteInteger.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros para obtener datos desde el terminal
input string             InpSymbolName="EURUSD";           // par de divisas
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;        // período de tiempo
input datetime           InpDateStart=D'2013.01.01 00:00'; // fecha de inicio del cop
//--- parámetros para la escritura de datos en el archivo
input string             InpFileName="Trend.bin"; // nombre del archivo
input string             InpDirectoryName="Data"; // nombre de la carpeta
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   double   close_buff[];
   datetime time_buff[];
   int      size;
//--- reseteamos el valor del error
   ResetLastError();
//--- copiamos el precio de cierre para cada barra
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-
     {
      PrintFormat("Fallo al copiar los valores de los precios de cierre. Código del e
      return;
     }
//--- copiamos el tiempo para cada barra
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Fallo al copiar el valor del tiempo. Código del error = %d",GetLas
      return;
     }
//--- obtenemos el tamaño del búfer
   size=ArraySize(close_buff);
//--- abrimos el archivo para escribir los valores (si no existe, se crea automáticam
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la escritura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //---
      int   up_down=0; // bandera de la tendencia
      int   arr_size;  // tamaño del array arr
      uchar arr[];     // array del tipo uchar
      //--- escribimos los valores de tiempo en el archivo
      for(int i=0;i<size-1;i++)
        {
         //--- comparamos los precios de cierre para la barra actual y la siguiente
         if(close_buff[i]<=close_buff[i+1])
           {
            if(up_down!=1)
              {
```

```
               //--- escribimos los valores de la fecha en el archivo utilizando File
               StringToCharArray(TimeToString(time_buff[i]),arr);
               arr_size=ArraySize(arr);
               //--- primero escribimos el número de símbolos del array
               FileWriteInteger(file_handle,arr_size,INT_VALUE);
               //--- escribimos los propios símbolos
               for(int j=0;j<arr_size;j++)
                  FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
               //--- cambiamos la bandera de la tendencia
               up_down=1;
              }
           }
         else
           {
            if(up_down!=-1)
              {
               //--- escribimos el valor de la fecha en el archivo utilizando FileWri
               StringToCharArray(TimeToString(time_buff[i]),arr);
               arr_size=ArraySize(arr);
               //--- primero escribimos el número de símbolos del array
               FileWriteInteger(file_handle,arr_size,INT_VALUE);
               //--- escribimos los propios símbolos
               for(int j=0;j<arr_size;j++)
                  FileWriteInteger(file_handle,arr[j],CHAR_VALUE);
               //--- cambiamos la bandera de la tendencia
               up_down=-1;
              }
           }
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos grabados, el archivo %s cerrado",InpFileName);
     }
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
  }
```

Véase también

IntegerToString, StringToInteger, Tipos enteros

FileWriteLong

Escribe el valor del parámetro del tipo long desde la posición actual del puntero de archivos en un archivo binario.

```
uint  FileWriteLong(
   int   file_handle,     // manejador del archivo
   long  value            // valor para escribir
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

value

[in] Valor del tipo long.

Valor devuelto

En caso del éxito la función devuelve el número de bytes escritos (en este caso sizeof(long)=8). El puntero de archivos se mueve a esta misma cantidad de bytes.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWriteLong.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros para obtener datos desde el terminal
input string          InpSymbolName="EURUSD";           // par de divisas
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // período de tiempo
input datetime        InpDateStart=D'2013.01.01 00:00'; // fecha de inicio del copiad
//--- parámetros para la escritura de datos en el archivo
input string          InpFileName="Volume.bin"; // nombre del archivo
input string          InpDirectoryName="Data";  // nombre de la carpeta
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish=TimeCurrent();
   long     volume_buff[];
   datetime time_buff[];
   int      size;
//--- reseteamos el valor del error
   ResetLastError();
//--- copiamos los volúmenes de ticks para cada barra
   if(CopyTickVolume(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,volume_bu
     {
      PrintFormat("Fallo al copiar los valores del volumen de ticks. Código del error
      return;
     }
//--- copiamos el tiempo para cada barra
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Fallo al copiar el valor del tiempo. Código del error = %d",GetLas
      return;
     }
//--- obtenemos el tamaño del búfer
   size=ArraySize(volume_buff);
//--- abrimos el archivo para escribir los valores del indicador (si no existe, se cr
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la escritura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- primero escribimos el tamaño de la muestra de datos
      FileWriteLong(file_handle,(long)size);
      //--- escribimos el tiempo y los valores del volumen en el archivo
      for(int i=0;i<size;i++)
        {
         FileWriteLong(file_handle,(long)time_buff[i]);
         FileWriteLong(file_handle,volume_buff[i]);
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos grabados, el archivo %s cerrado",InpFileName);
     }
```

```
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
}
```

Véase también

Tipos enteros, FileWriteInteger

FileWriteString

Esta función escribe el valor del parámetro del tipo string en un archivo del tipo BIN, CSV o TXT desde la posición actual del puntero de archivos. En el caso de la escritura en los archivos del tipo CSV o TXT: si en la cadena se encuentra el símbolo '\n' (LF) sin el símbolo '\r' (CR) que le precede, entonces antes del símbolo '\n' se añade el correspondiente símbolo '\r'.

```
uint  FileWriteString(
   int           file_handle,     // manejador del archivo
   const string  text_string,     // cadena para escribir
   int           length=-1        // número de símbolos
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

text_string

[in] Cadena.

length=-1

[in] Número de símbolos que hay que escribir. Este parámetro es necesario para escribir una cadena en el archivo del tipo BIN. Si el tamaño no está especificado, se escribe la cadena entera sin el 0 final. Si el tamaño especificado es menos que longitud de la cadena, se escribe una parte de la cadena sin 0 final. Si el tamaño especificado es más que longitud de la cadena, entonces la cadena se complementa con la cantidad de ceros correspondiente. Este parámetro se ignora para los archivos CSV y TXT y la cadena se escribe en su totalidad.

Valor devuelto

En caso del éxito la función devuelve el número de bytes escritos. El puntero de archivos se mueve a esta misma cantidad de bytes.

Nota

Hay que tener en cuenta que cuando escribimos en un archivo abierto con la bandera FILE_UNICODE (o sin bandera FILE_ANSI), el número de bytes escritos será doble del número de caracteres de cadena escritos. Cuando escribimos en un archivo abierto con la bandera FILE_ANSI, el número de bytes escritos va a coincidir con el número de símbolos de cadena que han sido escritos.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                        Demo_FileWriteString.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros para obtener datos desde el terminal
input string             InpSymbolName="EURUSD";           // par de divisas
input ENUM_TIMEFRAMES    InpSymbolPeriod=PERIOD_H1;        // período de tiempo
input int                InpMAPeriod=14;                   // período de la media móv
input ENUM_APPLIED_PRICE InpAppliedPrice=PRICE_CLOSE;      // tipo del precio
input datetime           InpDateStart=D'2013.01.01 00:00'; // fecha de inicio del cop
//--- parámetros para la escritura de datos en el archivo
input string             InpFileName="RSI.csv";    // nombre del archivo
input string             InpDirectoryName="Data"; // nombre de la carpeta
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   datetime date_finish; // fecha del fin de copiado de datos
   double   rsi_buff[];  // array de los valores del indicador
   datetime date_buff[]; // array de las fechas del indicador
   int      rsi_size=0;  // tamaño de arrays del indicador
//--- tiempo de finalización - actual
   date_finish=TimeCurrent();
//--- obtenemos el manejador del indicador RSI
   ResetLastError();
   int rsi_handle=iRSI(InpSymbolName,InpSymbolPeriod,InpMAPeriod,InpAppliedPrice);
   if(rsi_handle==INVALID_HANDLE)
     {
      //--- fallo al obtener el manejador del indicador
      PrintFormat("Error al obtener el manejador del indicador. Código del error = %d
      return;
     }
//--- seguimos en el ciclo hasta que el indicador no termine de calcular todos sus va
   while(BarsCalculated(rsi_handle)==-1)
      Sleep(10); // retardo para que al indicador le de tiempo a calcular sus valores
//--- copiamos los valores del indicador de un período determinado
   ResetLastError();
   if(CopyBuffer(rsi_handle,0,InpDateStart,date_finish,rsi_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores del indicador. Código del error = %d",
      return;
     }
//--- copiamos el tiempo correspondiente para los valores del indicador
   ResetLastError();
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,date_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores del tiempo. Código del error = %d",Get
      return;
     }
//--- liberamos la memoria que ocupa el indicador
   IndicatorRelease(rsi_handle);
//--- obtenemos el tamaño del búfer
   rsi_size=ArraySize(rsi_buff);
//--- abrimos el archivo para escribir los valores del indicador (si no existe, se cr
```

```
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la escritura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_DATA_PA
      //--- preparamos variables auxiliares
      string str="";
      bool   is_formed=false;
      //--- escribimos las fechas de formación de las zonas de sobrecompra y sobreven
      for(int i=0;i<rsi_size;i++)
        {
         //--- prueba de los valores del indicador
         if(rsi_buff[i]>=70 || rsi_buff[i]<=30)
           {
            //--- si es el primer valor en esta área
            if(!is_formed)
              {
               //--- añadimos el valor y la fecha
               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];
            //--- paso a la siguiente iteración del ciclo
            continue;
           }
         //--- prueba de la bandera
         if(is_formed)
           {
            //--- la cadena está formada, la escribimos en el archivo
            FileWriteString(file_handle,str+"\r\n");
            is_formed=false;
           }
        }
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos grabados, el archivo %s cerrado",InpFileName);
     }
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
  }
```

Véase también

Tip string, StringFormat

FileWriteStruct

Escribe el contenido de una estructura pasada como un parámetro en un archivo binario desde la posición actual del puntero de archivos.

```
uint  FileWriteStruct(
   int           file_handle,       // manejador del archivo
   const void&   struct_object,     // enlace a objeto
   int           size=-1            // tamaño para escribir en bytes
   );
```

Parámetros

file_handle

[in] Descriptor de archivo devuelto por la función FileOpen().

struct_object

[in] Referencia al objeto de dicha estructura. La estructura no debe contener cadenas, matrices dinámicas y funciones virtuales.

size=-1

[in] Número de símbolos que hay que escribir. Si el tamaño no está especificado o está indicada la cantidad de bytes que supera el tamaño de la estructura, se escribe la estructura entera.

Valor devuelto

En caso del éxito la función devuelve el número de bytes escritos. El puntero de archivos se mueve a esta misma cantidad de bytes.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                          Demo_FileWiteStruct.mq5 |
//|                        Copyright 2013, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2013, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
//--- mostramos la ventana de parámetros de entrada al iniciar el script
#property script_show_inputs
//--- parámetros para obtener datos desde el terminal
input string          InpSymbolName="EURUSD";           // par de divisas
input ENUM_TIMEFRAMES InpSymbolPeriod=PERIOD_H1;        // período de tiempo
input datetime        InpDateStart=D'2013.01.01 00:00'; // fecha de inicio del copiad
//--- parámetros para la escritura de datos en el archivo
input string          InpFileName="EURUSD.txt";         // nombre del archivo
input string          InpDirectoryName="Data";          // nombre de la carpeta
//+------------------------------------------------------------------+
//| Estructura para almacenar los datos de la vela                   |
//+------------------------------------------------------------------+
struct candlesticks
  {
   double            open;  // precio de apertura
   double            close; // precio de cierre
   double            high;  // precio máximo
   double            low;   // precio mínimo
   datetime          date;  // fecha
  };
//+------------------------------------------------------------------+
//| 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[];
//--- reseteamos el valor del error
   ResetLastError();
//--- obtenemos el tiempo de aparición de barras desde el rango
   if(CopyTime(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,time_buff)==-1)
     {
      PrintFormat("Fallo al copiar el valor de tiempo. Código del error = %d",GetLast
      return;
     }
//--- obtenemos los precios máximos de las barras desde el rango
   if(CopyHigh(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,high_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores de precios máximos. Código del error =
      return;
     }
//--- obtenemos los precios mínimos de las barras desde el rango
   if(CopyLow(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,low_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores de precios mínimos. Código del error =
      return;
     }
```

```
//--- obtenemos los precios de apertura de barras desde al rango
   if(CopyOpen(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,open_buff)==-1)
     {
      PrintFormat("Fallo al copiar los valores de precios de apertura. Código del err
      return;
     }
//--- obtenemos los precios de cierre de barras desde el rango
   if(CopyClose(InpSymbolName,InpSymbolPeriod,InpDateStart,date_finish,close_buff)==-
     {
      PrintFormat("Fallo al copiar los valores de precios de cierre. Código del error
      return;
     }
//--- definimos la dimensión de arrays
   size=ArraySize(time_buff);
//--- guardamos todos los datos en el array de la estructura
   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];
     }

//--- abrimos el archivo para la escritura del array de la estructura en el archivo (
   ResetLastError();
   int file_handle=FileOpen(InpDirectoryName+"//"+InpFileName,FILE_READ|FILE_WRITE|FI
   if(file_handle!=INVALID_HANDLE)
     {
      PrintFormat("El archivo %s está abierto para la escritura",InpFileName);
      PrintFormat("Ruta del archivo: %s\\Files\\",TerminalInfoString(TERMINAL_COMMOND
      //--- preparamos el contador del número de bytes
      uint counter=0;
      //--- escribimos los valores del array usando el ciclo
      for(int i=0;i<size;i++)
         counter+=FileWriteStruct(file_handle,cand_buff[i]);
      PrintFormat("En el archivo %s han sido escritos %d bytes de información",InpFil
      PrintFormat("Total de bytes: %d * %d * %d = %d, %s",size,5,8,size*5*8,size*5*8=
      //--- cerramos el archivo
      FileClose(file_handle);
      PrintFormat("Datos escritos, archivo %s cerrado",InpFileName);
     }
   else
      PrintFormat("Fallo al abrir el archivo %s, Código del error = %d",InpFileName,G
  }
```

Véase también

Estructuras y clases

FolderCreate

Crea una carpeta en el directorio Files (dependiendo del valor de common_flag)

```
bool  FolderCreate(
   string  folder_name,       // cadena con el nombre de la carpeta a crear
   int     common_flag=0      // zona de alcance
   );
```

Parámetros

folder_name

[in] Nombre de la carpeta que hay que crear. Contiene la ruta entera hacia la carpeta.

common_flag=0

[in] Bandera que determina la ubicación de la carpeta. Si es common_flag=FILE_COMMON, la carpeta se encuentra en la carpeta compartida de todos los terminales de cliente. De lo contrario, la carpeta se encuentra en la carpeta local (MQL5\files o MQL5\tester\files en caso de prueba).

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                              Demo_FolderCreate.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                     https://team.metaquotes.ru/email/view/599588 |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://team.metaquotes.ru/email/view/599588"
#property version   "1.00"
//--- descripción
#property description "El script muestra un ejemplo de utilización de FolderCreate().
#property description "Un parámetro externo define el directorio para crear carpetas.
#property description "Una vez ejecutado el script, se creará una estructura de carpe

//--- mostraremos la ventana de parámetros de entrada durante el arranque del script
#property script_show_inputs
//--- el parámetro de entrada define la carpeta en la que trabaja el script
input bool     common_folder=false; // la carpeta compartida para todos los terminale
int            flag=0;              // el valor de la bandera para determinar el luga
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string working_folder;
//--- estableceremos el valor de la bandera si el parámetro exteno common_folder==tru
   if(common_folder)
     {
      flag=FILE_COMMON;
      //--- buscaremos la carpeta en la que estamos trabajando
      working_folder=TerminalInfoString(TERMINAL_COMMONDATA_PATH)+"\\MQL5\\Files";
     }
   else working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
//--- la carpeta que vamos a crear en la carpeta MQL5\Files
   string root="Folder_A";
   if(CreateFolder(working_folder,root,flag))
     {
      //--- crearemos dentro la carpeta hija Child_Folder_B1
      string folder_B1="Child_Folder_B1";
      string path=root+"\\"+folder_B1;          // crearemos el nombre de la carpeta
      if(CreateFolder(working_folder,path,flag))
        {
         //--- en esta carpeta crearemos 3 carpetas hijas más
         string folder_C11="Child_Folder_C11";
         string child_path=path+"\\"+folder_C11;// crearemos el nombre de la carpeta
         CreateFolder(working_folder,child_path,flag);
         //--- la segunda carpeta hija
         string folder_C12="Child_Folder_C12";
         child_path=path+"\\"+folder_C12;
         CreateFolder(working_folder,child_path,flag);

         //--- la tercera carpeta hija
         string folder_C13="Child_Folder_C13";
         child_path=path+"\\"+folder_C13;
         CreateFolder(working_folder,child_path,flag);
        }
     }
//---
  }
//+------------------------------------------------------------------+
//|  intenta crear la carpeta y muestra los mensajes                 |
//+------------------------------------------------------------------+
```

```
bool CreateFolder(string working_folder,string folder_path,int file_flag)
  {
//--- mensaje de depuración
   PrintFormat("folder_path=%s",folder_path);
//--- intento de crear carpeta respecto a la ruta MQL5\Files
   if(FolderCreate(folder_path,file_flag))
     {
      //--- mostraremos la ruta entera para la carpeta creada
      PrintFormat("Hemos creado la carpeta %s",working_folder+"\\"+folder_path);
      //--- anularemos el código del error
      ResetLastError();
      //--- devolveremos el éxito
      return true;
     }
   else
      PrintFormat("Fallo al crear la carpeta %s. Código del error %d",working_folder+
//--- finalización fallida
   return false;
  }
```

Véase también

FileOpen(), FolderClean(), FileCopy()

FolderDelete

Elimina el directorio especificado. Si la carpeta no está vacía, no se puede eliminarla.

```
bool  FolderDelete(
   string  folder_name,       // cadena con el nombre de la carpeta a eliminar
   int     common_flag=0      // zona de alcance
   );
```

Parámetros

folder_name

[in] Nombre del directorio que hay que eliminar. Contiene la ruta entera hacia la carpeta.

common_flag=0

[in] Bandera que determina la ubicación de la carpeta. Si es common_flag=FILE_COMMON, la carpeta se encuentra en la carpeta compartida de todos los terminales de cliente. De lo contrario, la carpeta se encuentra en la carpeta local (MQL5\files o MQL5\tester\files en caso de prueba).

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

Si el directorio contiene al menos un archivo y/o un subdirectorio, es imposible eliminar este directorio; previamente hay que desocuparlo. La liberación total de una carpeta de todos los archivos y subcarpetas anidadas se realiza mediante la función FolderClean().

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                              Demo_FolderDelete.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                      https://team.metaquotes.ru/email/view/599588 |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://team.metaquotes.ru/email/view/599588"
#property version   "1.00"
//--- descripción
#property description "El script muestra el ejemplo del uso de FolderDelete()."
#property description "Primero se crean dos carpetas: una está vacía, la otra contien
#property description "Cuando se intenta eliminar la carpeta no vacía, se devuelve el

//--- mostraremos la ventana de los parámetros de entrada durante el inicio del scrip
#property script_show_inputs
//--- parámetros de entrada
input string   firstFolder="empty";     // carpeta vacía
input string   secondFolder="nonempty";// carpeta que va a contener un archivo
string filename="delete_me.txt";       // nombre del archivo que vamos a crear en la
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- escribiremos el manejador del archivo aquí
   int handle;
//--- buscaremos la carpeta en la que trabajamos
   string working_folder=TerminalInfoString(TERMINAL_DATA_PATH)+"\\MQL5\\Files";
//--- mensaje de depuración
   PrintFormat("working_folder=%s",working_folder);
//--- intento de crear una carpeta vacía respecto a la ruta MQL5\Files
   if(FolderCreate(firstFolder,0)) // 0 significa que estamos trabajando en la carpet
     {
      //--- mostraremos la ruta completa de la carpeta creada
      PrintFormat("Hemos creado la carpeta %s",working_folder+"\\"+firstFolder);
      //--- anularemos el código del error
      ResetLastError();
     }
   else
      PrintFormat("Fallo al crear la carpeta %s. Código del error %d",working_folder+

//--- ahora crearemos una carpeta no vacía usando la función FileOpen()
   string filepath=secondFolder+"\\"+filename;  // formaremos la ruta para el archivo
   handle=FileOpen(filepath,FILE_WRITE|FILE_TXT); // la bandera FILE_WRITE es obligat
   if(handle!=INVALID_HANDLE)
      PrintFormat("El archivo %s ha sido abierto para la lectura",working_folder+"\\"
   else
      PrintFormat("Fallo al crear el archivo %s en la carpeta %s. Código del error=",

   Comment(StringFormat("Preparándose para eliminar las carpetas %s y %s", firstFolde
//--- Una pequeña pausa de 5 segundos para que nos de tiempo a leer el mensaje en el
   Sleep(5000); // ¡No se puede usar Sleep() en los indicadores!

//--- mostramos la ventana de diálogo y pedimos al usuario
   int choice=MessageBox(StringFormat("¿Desea eliminar las carpetas %s y %s?", firstF
                         "Eliminando carpetas",
                         MB_YESNO|MB_ICONQUESTION); //  habrá dos botones - "Yes" y "

//--- ejecutaremos la acción en función de la opción escogida
   if(choice==IDYES)
     {
```

```
      //--- quitamos el comentario del gráfico
      Comment("");
      //--- mostraremos el mensaje en el diario "Asesores Expertos"
      PrintFormat("Intentamos eliminar las carpetas %s y %s",firstFolder, secondFolde
      ResetLastError();
      //--- eliminamos la carpeta vacía
      if(FolderDelete(firstFolder))
         //--- debe aparecer el siguiente mensaje porque la carpeta está vacía
         PrintFormat("La carpeta %s ha sido eliminada con éxito",firstFolder);
      else
         PrintFormat("Fallo al eliminar la carpeta %s. Código del error=%d", firstFol

      ResetLastError();
      //--- eliminamos la carpeta que contiene el archivo
      if(FolderDelete(secondFolder))
         PrintFormat("La carpeta %s ha sido eliminada con éxito", secondFolder);
      else
         //--- debe aparecer este mensaje porque hay un archivo dentro de la carpeta
         PrintFormat("Fallo al eliminar la carpeta %s. Código del error=%d", secondFo
     }
   else
      Print("Eliminación cancelada");
//---
  }
```

Véase también

FileOpen(), FolderClean(), FileMove()

FolderClean

Elimina todos los archivos en una carpeta especificada.

```
bool  FolderClean(
   string  folder_name,       // cadena con el nombre de subcarpeta
   int     common_flag=0      // zona de alcance
   );
```

Parámetros

folder_name

[in] Nombre del directorio donde hay que eliminar todos los archivos. Contiene la ruta entera hacia la carpeta.

common_flag=0

[in] Bandera que determina la ubicación de la carpeta. Si es common_flag=FILE_COMMON, la carpeta se encuentra en la carpeta compartida de todos los terminales de cliente. De lo contrario, la carpeta se encuentra en la carpeta local (MQL5\files o MQL5\tester\files en caso de prueba).

Valor devuelto

Devuelve true en caso de éxito, de lo contrario devuelve false.

Nota

Cuidado con usar esta función porque todos los archivos y todos los subdirectorios anidados se eliminan sin que se pueda recuperarlos.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_FolderClean.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                      https://team.metaquotes.ru/email/view/599588 |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "https://team.metaquotes.ru/email/view/599588"
#property version   "1.00"
//--- descripción
#property description "Este script muestra un ejemplo de uso de FolderClean()."
#property description "Primero se crean los archivos en la carpeta especificada utili
#property description "Luego, antes de eliminar los archivos, se muestra el aviso uti

//--- mostraremos la ventana de los parámetros de entrada durante el inicio del scrip
#property script_show_inputs
//--- parámetros de entrada
input string   foldername="demo_folder";  // crearemos la carpeta en MQL5/Files/
input int      files=5;                   // número de archivos que vamos a crear y e
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
   string name="testfile";
//--- primero abrimos o creamos archivos en la carpeta de datos de nuestro terminal
   for(int N=0;N<files;N++)
     {
      //--- nombre del archivo como 'demo_folder\testfileN.txt'
      string filemane=StringFormat("%s\\%s%d.txt",foldername,name,N);
      //--- abrimos el archivo con la bandera para escribir, en este caso la carpeta
      int handle=FileOpen(filemane,FILE_WRITE);
      //--- veremos si la función FileOpen() ha trabajado con éxito
      if(handle==INVALID_HANDLE)
        {
         PrintFormat("Fallo al crear el archivo %s. Código del error ",filemane,GetLa
         ResetLastError();
        }
      else
        {
         PrintFormat("El archivo %s ha sido abierto con éxito",filemane);
         //--- ya no necesitamos el archivo abierto, hay que cerrarlo sí o sí
         FileClose(handle);
        }
     }

//--- comprobaremos el número de archivos en la carpeta
   int k=FilesInFolder(foldername+"\\*.*",0);
   PrintFormat("En total, la carpeta %s contiene %d archivos",foldername,k);

//--- mostramos la ventana de diálogo y preguntamos al usuario
   int choice=MessageBox(StringFormat("¿Desea eliminar de la carpeta %s %d archivos,
                         "Eliminando archivos de la carpeta",
                         MB_YESNO|MB_ICONQUESTION); //  habrá dos botones - "Yes" y "
   ResetLastError();

//--- ejecutaremos acciones en función de la opción seleccionada
   if(choice==IDYES)
     {
      //--- empezamos a eliminar
      PrintFormat("Intento de eliminar todos los archivos de la carpeta %s",foldernam
      if(FolderClean(foldername,0))
```

```
         PrintFormat("Archivos eliminados con éxito, en la carpeta %s quedan %d archi
                     foldername,
                     FilesInFolder(foldername+"\\*.*",0));
      else
         PrintFormat("Fallo al eliminar los archivos de la carpeta %s. Código del err
     }
   else
      PrintFormat("Eliminación cancelada");
//---
  }
//+------------------------------------------------------------------+
//| devuelve el número de archivos en la carpeta especificada        |
//+------------------------------------------------------------------+
int FilesInFolder(string path,int flag)
  {
   int count=0;
   long handle;
   string filename;
//---
   handle=FileFindFirst(path,filename,flag);
//--- si ha sido encontrado por lo menos un archivo, seguimos buscando el resto
   if(handle!=INVALID_HANDLE)
     {
      //--- mostraremos el nombre del archivo
      PrintFormat("ha sido encntrado el archivo %s",filename);
      //--- aumentaremos el contador de archivos/carpetas encontrados
      count++;
      //--- iniciamos la búsqueda en todos los archivos/carpetas
      while(FileFindNext(handle,filename))
        {
         PrintFormat("ha sido encontrado el archivo %s",filename);
         count++;
        }
      //--- es obligatorio cerrar el manejador del buscador al finalizar
      FileFindClose(handle);
     }
   else // fallo al obtener el manejador
     {
      PrintFormat("Fallo al realizar la búsqueda de archivos en la carpeta %s",path);
     }
//--- devolveremos el resultado
   return count;
  }
```

Véase también

FileFindFirst, FileFindNext, FileFindClose

Indicadores personalizados

Grupo de funciones que se usan para crear los indicadores personalizados. No se puede usarlas para crear los Asesores Expertos y los scripts.

Función	Acción
SetIndexBuffer	Enlaza el búfer de indicadores especificado con el array dinámico unidimensional del tipo double
IndicatorSetDouble	Establece el valor de una propiedad del indicador que tiene el tipo double
IndicatorSetInteger	Establece el valor de una propiedad del indicador que tiene el tipo int
IndicatorSetString	Establece el valor de una propiedad del indicador que tiene el tipo string
PlotIndexSetDouble	Establece el valor de propiedad de línea del indicador que tiene el tipo double
PlotIndexSetInteger	Establece el valor de propiedad de línea del indicador que tiene el tipo int
PlotIndexSetString	Establece el valor de propiedad de línea del indicador que tiene el tipo string
PlotIndexGetInteger	Devuelve el valor de propiedad de línea del indicador que tiene el tipo entero

Las propiedades de los indicadores se puede establecer tanto utilizando las directivas del compilador, como a través de las funciones. Para el mejor entendimiento del asunto, se recomienda estudiar los estilos de los indicadores en los ejemplos.

Todos los cálculos necesarios de los indicadores personalizados hay que colocar en la función predeterminada OnCalculate(). Si se usa la forma breve de la llamada a la función OnCalculate() del tipo

```
int OnCalculate (const int rates_total, const int prev_calculated, const int begin, c
```

entonces la variable *rates_total* contiene el valor de la cantidad total de elementos del array price[], que ha sido pasado como parámetro input para calcular valores del indicador.

El parámetro *prev_calculated* es el resultado de ejecución de la función OnCalculate() en la llamada anterior, permite organizar un algoritmo económico para calcular valores del indicador. Por ejemplo, si el valor actual rates_total=1000, y prev_calculated=999, entonces es posible que sea suficiente hacer los cálculos sólo para un valor de cada búfer de indicadores.

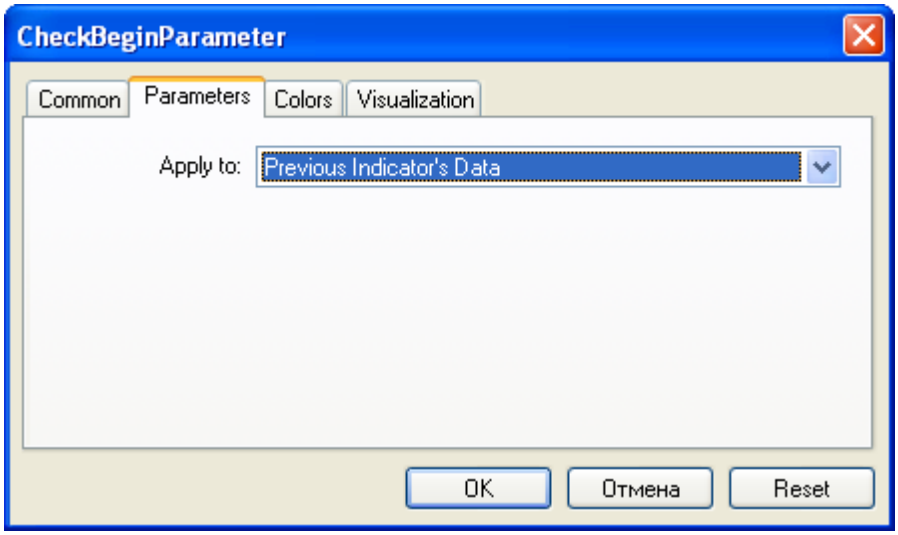
Si la información sobre el tamaño del array de entrada price no estuviera disponible, sería necesario hacer los cálculos para 1000 valores de cada búfer de indicador. Con la primera llamada a la función OnCalculate() el valor prev_calculated=0. Si de alguna manera el array price[] ha sido cambiado, en este caso prev_calculated también es igual a 0.

El parámetro *begin* dice el número de valores iniciales del array price que no contienen los datos para el cálculo. Por ejemplo, si los valores del indicador Accelerator Oscillator (para el que los primeros 37 valores no se calculan) han sido usados como un parámetro de entrada, entonces begin=37. Como ejemplo vamos a ver un indicador simple:

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   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 int begin,
                const double &price[])

  {
//---
   Print("begin = ",begin,"  prev_calculated = ",prev_calculated,"  rates_total = ",r
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Vamos a arrastrarlo desde la ventana "Navegador" a la ventana del indicador Accelerator Oscillator e indiquemos que los cálculos serán realizados a base de los valores del indicador anterior:



Como resultado, con la primera llamada a la función OnCalculate() el valor prev_calculated va a ser igual a cero, y con las siguientes llamadas será igual al valor rates_total (hasta que el número de barras en el gráfico de precios no se aumente).

El valor del parámetro begin será exactamente igual al número de barras iniciales para las que los valores del indicador Accelerator no se calculan conforme a la lógica de este indicador. Si nos fijamos en el código fuente del indicador personalizado Accelerator.mq5, veremos en la función OnInit() estas líneas:

```
//--- sets first bar from what index will be drawn
   PlotIndexSetInteger(0,PLOT_DRAW_BEGIN,37);
```

Precisamente usando la función PlotIndexSetInteger(0, PLOT_DRAW_BEGIN, empty_first_values) comunicamos el número de primeros valores inexistentes en el array de indicador nulo del indicador

personalizado, los que no tenemos que considerar para el cálculo (empty_first_values). De esta manera, disponemos de mecanismos para:

1. comunicar sobre el número de valores iniciales de un indicador los que no hay que usar para los cálculos en otro indicador personalizado;
2. obtener la información sobre el número de primeros valores iniciales los que hay que ignorar a la hora de llamar a otro indicador personalizado sin entrar en la lógica de sus cálculos.

Estilos de indicadores en ejemplos

El terminal de cliente MetaTrader 5 cuenta con 38 indicadores técnicos incorporados que se puede utilizar en los programas MQL5 a través de las funciones correspondientes. Pero el principal mérito del lenguaje MQL5 consiste en la posibilidad de crear sus propios indicadores personalizados que luego pueden ser utilizados en los Asesores Expertos para obtener valores, o simplemente pueden ser aplicados a los gráficos de precios para realizar el análisis técnico.

Usted puede conseguir toda la variedad de indicadores a base de unos estilos de dibujo básicos, llamados construcciones gráficas. Bajo la construcción se entiende el modo de visualización de datos que el indicador calcula, guarda y ofrece si se solicitan. Hay siete tipos de construcciones básicas:

1. línea,
2. sección (segmento),
3. histograma,
4. flecha (símbolo),
5. área coloreada (canal con relleno),
6. barras,
7. velas japonesas.

Cada construcción requiere para su visualización de uno a cinco arrays del tipo double en los que se guardan los valores del indicador. Para un cómodo trabajo del indicador, estos arrays se asocian a los búfers indicadores. La cantidad de búfers en el indicador hace falta declarar de antemano, utilizando las directivas del compilador, por ejemplo:

```
#property indicator_buffers 3 // número de búfers
#property indicator_plots   2 // número de construcciones gráficas
```

El número de búfers en el indicador siempre es igual o superior al número de construcciones gráficas en el indicador.

Puesto que cada construcción gráfica base puede tener diferentes variaciones de colores o un carácter específico de visualización, el número real de construcciones en el lenguaje MQL5 es 18:

Construcción	Descripción	Búfers de valores	Búfers de colores
DRAW_NONE	No se muestra visualmente en el gráfico, pero se puede ver los valores del búfer correspondiente en la "Ventana de datos"	1	-
DRAW_LINE	Se traza una línea a base de los valores del búfer correspondiente (los valores vacíos son indeseables en el búfer)	1	-

DRAW_SECTION	Se traza como una línea segmentada entre los valores del búfer correspondiente (normalmente hay muchos valores vacíos)	1	-
DRAW_HISTOGRAM	Se traza como un histograma desde la línea cero hasta los valores del búfer correspondiente (puede tener valores vacíos)	1	-
DRAW_HISTOGRAM2	Se traza como un histograma basándose en dos búfers indicadores (puede tener valores vacíos)	2	-
DRAW_ARROW	Se traza como símbolos (puede tener valores vacíos)	1	-
DRAW_ZIGZAG	Parecido al estilo DRAW_SECTION, pero a diferencia de éste puede dibujar segmentos verticales en una barra	2	-
DRAW_FILLING	Relleno de colores entre dos líneas. En la "Ventana de datos" se muestran 2 valores de los búfers correspondientes	2	-
DRAW_BARS	Se visualiza en el gráfico en forma de las barras. En la "Ventana de datos" se muestran 4 valores de los búfers correspondientes	4	-
DRAW_CANDLES	Se visualiza en forma de las velas japonesas. En la "Ventana de datos" se muestran 4 valores de los búfers	4	-

	correspondientes		
DRAW_COLOR_LINE	Es una línea para la que se puede alternar colores en diferentes barras y cambiar su calor en el momento deseado	1	1
DRAW_COLOR_SECTION	Parecido al estilo DRAW_SECTION, pero el color para cada sección se puede definir de manera individual; también se puede definir el color de forma dinámica	1	1
DRAW_COLOR_HISTOGRAM	Parecido al estilo DRAW_HISTOGRAM, pero cada raya puede tener su propio color; también se puede definir el color de forma dinámica	1	1
DRAW_COLOR_HISTOGRAM2	Parecido a DRAW_HISTOGRAM2, ero cada raya puede tener su propio color; también se puede definir el color de forma dinámica	2	1
DRAW_COLOR_ARROW	Parecido al estilo DRAW_ARROW, pero cada símbolo puede tener su propio color. Se puede cambiar el color dinámicamente	1	1
DRAW_COLOR_ZIGZAG	El estilo DRAW_ZIGZAG con las posibilidades del coloreado individual de secciones y el cambio dinámico del color	2	1
DRAW_COLOR_BARS	Estilo DRAW_BARS con las posibilidades del coloreado individual de barras y el cambio dinámico	4	1

	del color		
DRAW_COLOR_CANDLES	Estilo DRAW_CANDLES con las posibilidades del coloreado individual de velas y el cambio dinámico del color	4	1

La diferencia entre un búfer indicador y un array

En cada indicador hay que declarar a un nivel global uno o más arrays del tipo double que luego tendrá que ser utilizado como búfer indicador mediante la función SetIndexBuffer(). Para dibujar las construcciones gráficas del indicador, se utilizan sólo los valores desde el búfer indicador. No se puede utilizar ningún otro array para este propósito. Además de eso, los valores de los búfers se muestran en la "Ventana de datos".

Un búfer indicador tiene que ser dinámico y no requiere la especificación del tamaño – el tamaño del array que ha sido utilizado como búfer indicador se determina por el subsistema ejecutable del terminal de forma automática.

Tras la vinculación del array con el búfer indicador, la dirección de indexación se establece por defecto como en los arrays comunes, pero si hace falta, Usted puede aplicar la función ArraySetAsSeries() para cambiar el modo de acceso a los elementos del array. Por defecto, el búfer indicador se utiliza para almacenar los datos que están destinados para el proceso de dibujo (INDICATOR_DATA).

Si para el cálculo de los valores del indicador es necesario realizar algunas computaciones intermedias y almacenar un valor adicional para cada barra, entonces durante la vinculación este array puede ser declarado como el búfer de cálculo (INDICATOR_CALCULATIONS). Un array ordinario también puede ser utilizado para los valores intermedios, pero en este caso el programador debe encargarse personalmente de controlar el tamaño de este array.

Algunas construcciones permiten establecer un color de visualización para cada barra. Para almacenar la información sobre el color, se utilizan los búfers de colores (INDICATOR_COLOR_INDEX). El color está representado con el tipo de números enteros color, pero todos los búfers indicadores deben tener el tipo double. No se puede obtener los valores de colores y de los búfers auxiliares (INDICATOR_CALCULATIONS) mediante la función CopyBuffer().

El número de los búfers indicadores debe ser indicado por la directiva del compilador #property indicator_buffers número_de_búfers:

```
#property indicator_buffers 3  //  el indicador tiene 3 búfers
```

El número máximo permitido de los búfers en un indicador - 512.

Correspondencia de búfers indicadores y construcciones gráficas

Cada construcción gráfica se basa en uno o más búfers indicadores. De esta manera, para visualizar las velas japonesas simples, hacen falta cuatro valores: los precios Open, High, Low y Close. En

consecuencia, para visualizar un indicador en forma de velas japonesas, es necesario declarar 4 búfers indicadores y 4 arrays del tipo double para ellos. Por ejemplo:

```
//--- el indicador tiene cuatro búfers indicadores
#property indicator_buffers 4
//--- el indicador tiene una construcción gráfica
#property indicator_plots   1
//--- la construcción gráfica con el número 1 se mostrará en forma de velas japonesas
#property indicator_type1   DRAW_CANDLES
//--- las velas japonesas serán dibujadas en color clrDodgerBlue
#property indicator_color1  clrDodgerBlue
//--- 4 arrays para los búfers indicadores
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];
```

Las construcciones gráficas utilizan automáticamente los búfers indicadores de acuerdo con el número de construcción. Los números de construcción se empiezan desde 1, los números de búfers se empiezan desde 0. Si la primera construcción requiere 4 búfers indicadores, para el trazado serán utilizados los 4 primeros búfers indicadores. Estos cuatro búfers indicadores deben estar vinculados con los arrays correspondientes con la indexación correcta mediante la función SetIndexBuffer().

```
//--- Vinculación de arrays con búfers indicadores
   SetIndexBuffer(0,OBuffer,INDICATOR_DATA);  // el primer búfer corresponde al índic
   SetIndexBuffer(1,HBuffer,INDICATOR_DATA);  // el segundo búfer corresponde al índi
   SetIndexBuffer(2,LBuffer,INDICATOR_DATA);  // el tercer búfer corresponde al índic
   SetIndexBuffer(3,CBuffer,INDICATOR_DATA);  // el cuarto búfer corresponde al índic
```

Durante el dibujo de velas japonesas el indicador va a utilizar precisamente los cuatro primeros búfers, porque la construcción "velas japonesas" ha sido declarada bajo el número uno.

Vamos a cambiar un poco nuestro ejemplo, vamos a añadir la construcción en forma de una línea simple - DRAW_LINE. Ahora, que la línea tenga el número 1, y las velas japonesas el número 2. El número de búfers y el número de construcciones se ha aumentado.

```
//--- el indicador tiene 5 búfers indicadores
#property indicator_buffers 5
//--- el indicador tiene 2 construcciones gráficas
#property indicator_plots   2
//--- la construcción gráfica con el número 1 se mostrará en forma de la línea
#property indicator_type1   DRAW_LINE
//--- la línea tendrá el color clrDodgerRed
#property indicator_color1  clrDodgerRed
//---la construcción gráfica con el número 2 se mostrará en forma de velas japonesas
#property indicator_type2   DRAW_CANDLES
//--- las velas japonesas serán dibujadas en color clrDodgerBlue
#property indicator_color2  clrDodgerBlue
//--- 5 arrays para los búfers indicadores
```

```
double LineBuffer[];
double OBuffer[];
double HBuffer[];
double LBuffer[];
double CBuffer[];
```

El orden de construcciones se ha cambiado, ahora primero va la línea, luego van las velas japonesas. Por esta razón, el orden de búfers va a ser el apropiado. Es decir, primero declararemos bajo el índice cero el búfer para la línea, y luego cuatro búfers para la visualización de velas japonesas.

```
   SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);  // el primer búfer corresponde al ín
//--- vinculación de arrays con búfers indicadores para las velas japonesas
   SetIndexBuffer(1,OBuffer,INDICATOR_DATA);     // el segundo búfer corresponde al í
   SetIndexBuffer(2,HBuffer,INDICATOR_DATA);     // el tercer búfer corresponde al ín
   SetIndexBuffer(3,LBuffer,INDICATOR_DATA);     // el cuarto búfer corresponde al ín
   SetIndexBuffer(4,CBuffer,INDICATOR_DATA);     // el quinto búfer corresponde al ín
```

El número de búfers y construcciones gráficas se puede fijar sólo a través las directivas del compilador, es imposible el cambio dinámico de estas propiedades a través de las funciones.

Versiones de color de los estilos

Como se puede divisar de la tabla, los estilos se dividen en dos grupos. Al primer grupo pertenecen los estilos en cuyos nombres no figura la palabra **COLOR**, vamos a llamarlos estilos básicos:

- DRAW_LINE
- DRAW_SECTION
- DRAW_HISTOGRAM
- DRAW_HISTOGRAM2
- DRAW_ARROW
- DRAW_ZIGZAG
- DRAW_FILLING
- DRAW_BARS
- DRAW_CANDLES

El segundo grupo de estilos contiene la palabra **COLOR**, vamos a llamarlos versiones de color:

- DRAW_COLOR_LINE
- DRAW_COLOR_SECTION
- DRAW_COLOR_HISTOGRAM
- DRAW_COLOR_HISTOGRAM2
- DRAW_COLOR_ARROW
- DRAW_COLOR_ZIGZAG
- DRAW_COLOR_BARS
- DRAW_COLOR_CANDLES

Todas las versiones de color de los estilos se diferencian de los temas básicos, lo que permite especificar un color para cada parte de la construcción. La parte mínima de la construcción es una barra, por eso se puede decir que las versiones de color permiten especificar un color para cada barra.

Para que se pueda especificar el color de la construcción en cada barra, en las versiones de color de los estilos ha sido agregado un búfer adicional especial para almacenar el índice del color. Estos índices indican el número del color en un array especial que contiene el conjunto de colores especificado de antemano. El tamaño del array de los colores - 64. Esto quiere decir que cada versión de color del estilo permite colorear una construcción gráfica con 64 diferentes colores.

El conjunto y el número de colores en este array especial se puede establecer con la directiva del compilador #property indicator_color, donde Usted puede especificar todos los colores necesarios separados con coma. Por ejemplo, hacemos la siguiente entrada en el indicador:

```
//--- estableceremos 8 colores para colorear las velas (se guardan en un array especi
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
```

Aquí está indicado que para la construcción gráfica número 1 hemos establecido 8 colores que serán colocados en el array especial. Luego en el programa no vamos a indicar el color para la construcción gráfica, sino vamos a utilizar su índice. Si queremos establecer para una barra el color rojo,para ello es necesario indicar en el búfer de colores el índice del color rojo desde el array. En la directiva el rojo va el primero, le corresponde el índice con el número 0.

```
   //--- fijaremos el color de la vela clrRed
   col_buffer[buffer_index]=0;
```

El conjunto de colores no es algo determinado de una vez y para siempre, se puede cambiarlo dinámicamente a través de la función PlotIndexSetInteger(). Ejemplo:

```
      //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
      PlotIndexSetInteger(0,                    //  número del estilo gráfico
                          PLOT_LINE_COLOR,      //  identificador de la propiedad
                          plot_color_ind,       //  índice del color donde escribirem
                          color_array[i]);      //  nuevo color
```

Propiedades del indicador y construcciones gráficas

Para las construcciones gráficas se puede establecer las propiedades tanto mediante las directivas del compilador, como a través de las funciones correspondientes. En el apartado Relación entre las propiedades de indicador y funciones correspondientes este asunto se describe con más detalles. El cambio dinámico de las propiedades del indicador a través de las funciones permite crear los indicadores personalizados más flexibles.

Inicio del proceso de dibujo de un indicador en el gráfico

En muchas ocasiones, según las condiciones del algoritmo, resulta imposible iniciar el cálculo de los valores del indicador inmediatamente desde la barra actual, es necesario proporcionar el número mínimo de las barras previas disponibles en el historial. Por ejemplo, muchos tipos del suavizado suponen el uso del array de precios para N barras anteriores, y a base de estos valores se calcula el

valor del indicador para la barra actual.

En estas ocasiones, o no hay posibilidad de calcular los valores del indicador en las primeras N barras, o estos valores no están destinados para ser visualizados en el gráfico y conllevan un papel auxiliar para el cálculo de siguientes valores. Para renunciar la visualización del indicador en las primeras N barras del historial, se debe establecer el valor N para la propiedad PLOT_DRAW_BEGIN para la construcción gráfica correspondiente:

```
//--- vinculación de arrays con búfers indicadores para las velas japonesas
PlotIndexSetInteger(número_de_construcción_gráfica,PLOT_DRAW_BEGIN,N);
```

Aquí:

- número_de_construcción_gráfica - un valor de cero a indicator_plots-1 (la numeración de construcciones gráficas se empieza desde cero).
- N - número de las primeras barras en el historial en las cuales el indicador no debe mostrarse en el gráfico.

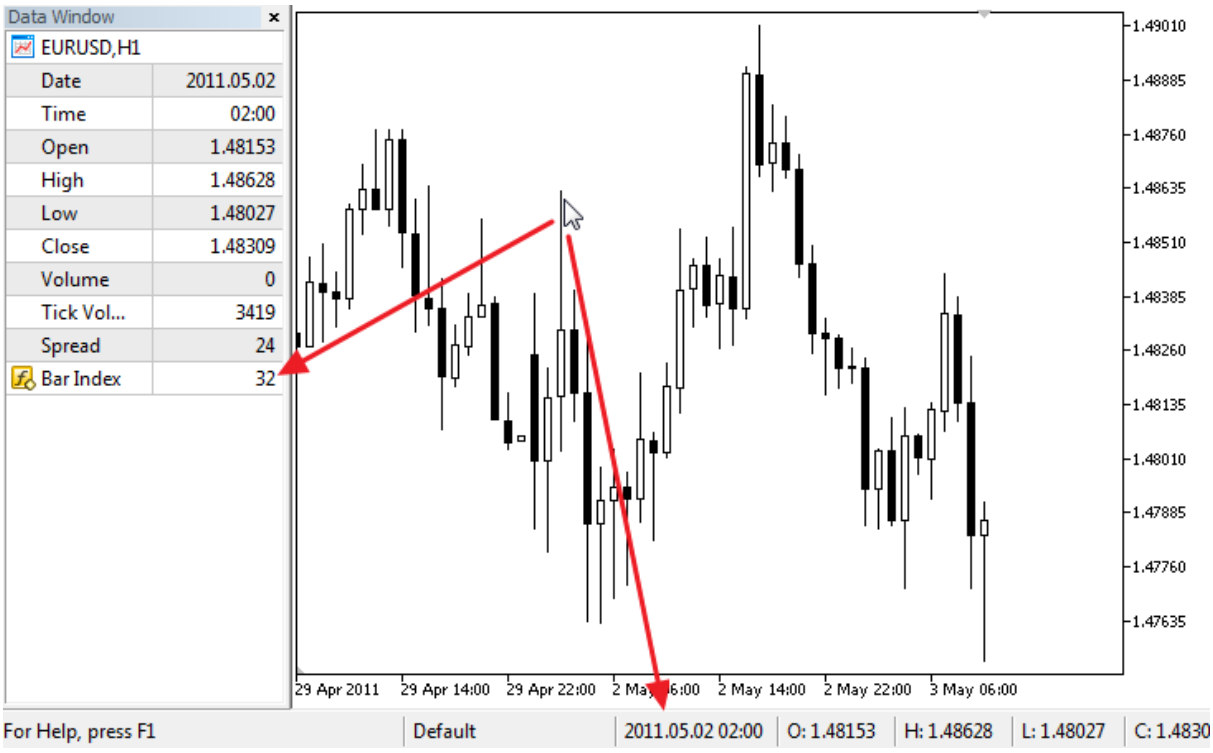
DRAW_NONE

El estilo DRAW_NONE se utiliza cuando es necesario calcular los valores del búfer y mostrarlos en la "Ventana de datos", pero la misma representación en el gráfico no se precisa. Para ajustar la precisión de representación, utilice la expresión IndicatorSetInteger(INDICATOR_DIGITS,número_de_dígitos) en la función OnInit():

```
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
//--- fijamos la precisión para el valor a mostrar en la Ventana de datos
   IndicatorSetInteger(INDICATOR_DIGITS,0);
//---
   return(INIT_SUCCEEDED);
  }
```

El número de búfers requeridos para construir DRAW_NONE — 1.

Aquí tenemos un ejemplo del indicador que muestra en la "Ventana de datos" el número de la barra sobre la que está situado el ratón. La numeración corresponde a la serie temporal. Es decir, la barra actual no finalizada tiene el índice cero, mientras que la barra más antigua tiene el índice más grande.



Fíjense que a pesar de que la construcción gráfica №1 tenga especificado el color rojo de representación, el indicador no dibuja nada en el gráfico.

```
//+------------------------------------------------------------------+
//|                                                    DRAW_NONE.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
```

```
//|                                                 http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Invisible
#property indicator_label1  "Bar Index"
#property indicator_type1   DRAW_NONE
#property indicator_style1  STYLE_SOLID
#property indicator_color1  clrRed
#property indicator_width1  1
//--- indicator buffers
double         InvisibleBuffer[];
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array con el búfer indicador
   SetIndexBuffer(0,InvisibleBuffer,INDICATOR_DATA);
//--- fijaremos la precisión con la que el valor será mostrado en la "Ventana de dato
   IndicatorSetInteger(INDICATOR_DIGITS,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[])
  {
   static datetime lastbar=0;
//--- si es el primer cálculo del indicador,
   if(prev_calculated==0)
     {
      //--- reenumeraremos las barras por primera vez
      CalcValues(rates_total,close);
```

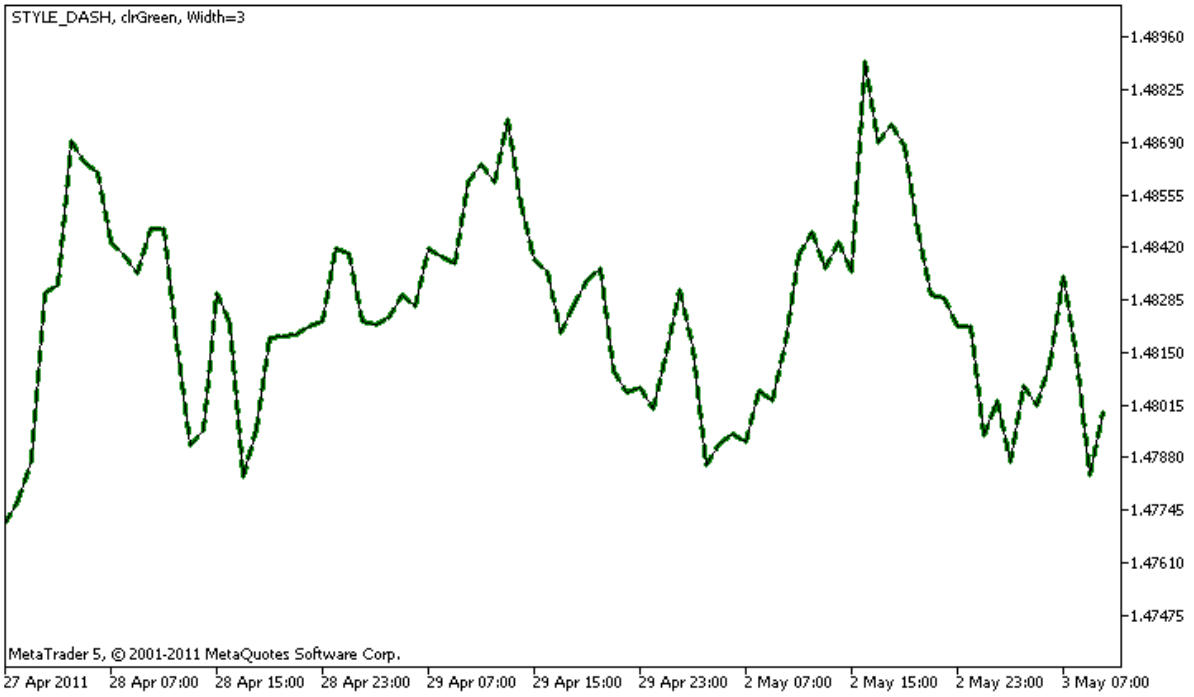
```
      //--- recordaremos la hora de apertura de la barra actual en lastbar
      lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
     }
   else
     {
      //--- si ha aparecido una barra nueva, la hora de su apertura no coincide con l
      if(lastbar!=SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE))
        {
         //--- volveremos a enumerar las barras de nuevo
         CalcValues(rates_total,close);
         //--- actualizaremos la hora de apertura de la barra actual en lastbar
         lastbar=(datetime)SeriesInfoInteger(_Symbol,_Period,SERIES_LASTBAR_DATE);
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| enumeramos las barras como en una serie temporal                 |
//+------------------------------------------------------------------+
void CalcValues(int total,double const  &array[])
  {
//--- estableceremos para el búfer indicador la indexación como en la serie temporal
   ArraySetAsSeries(InvisibleBuffer,true);
//--- llenaremos para cada barra su número
   for(int i=0;i<total;i++) InvisibleBuffer[i]=i;
  }
```

DRAW_LINE

El estilo DRAW_LINE dibuja una línea de color especificado a base de los valores del búfer indicador. Usted puede establecer el grosor, color y el estilo de la línea tanto con las directivas del compilador, como de forma dinámica, utilizando la función PlotIndexSetInteger(). La opción del cambio dinámico de las propiedades de la construcción gráfica permite crear indicadores "vivos", es decir, los que cambian su apariencia en función de la situación actual.

El número de búfers requeridos para construir DRAW_LINE — 1.

Aquí tenemos un ejemplo de un indicador que traza una línea usando los precios de cierre de las barras Close. El color, grosor y el estilo de la línea se cambian de forma aleatoria cada N=5 tics.



Tenga presente que inicialmente para la construcción gráfica **plot1** con el estilo DRAW_LINE las propiedades se establecen mediante la directiva del compilador #property, y luego en la función OnCalculate() estas tres propiedades se establecen de forma aleatoria. El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                                    DRAW_LINE.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_LINE"
#property description "Dibuja una línea de color especificado a base de los precios C
#property description "El color, grosor y el estilo de la línea se cambian de forma a
```

```
#property description "dentro de cada N tics"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- las propiedades de la línea están establecidas por medio de las directivas del
#property indicator_label1  "Line"      // nombre de la construcción para la "Ventana
#property indicator_type1   DRAW_LINE   // tipo de construcción gráfica - una línea
#property indicator_color1  clrRed      // color de la línea
#property indicator_style1  STYLE_SOLID // estilo de la línea
#property indicator_width1  1           // grosor de la línea
//--- parámetro input
input int      N=5;          // número de tics para el cambio
//--- búfer indicador para la construcción
double         LineBuffer[];
//--- array para almacenar colores
color colors[]={clrRed,clrBlue,clrGreen};
//--- array para almacenar estilos de trazado de la línea
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array con el búfer indicador
   SetIndexBuffer(0,LineBuffer,INDICATOR_DATA);
//--- inicialización del generador de números pseudoaleatorios
   MathSrand(GetTickCount());
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
```

```
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- bloque para calcular los valores del indicador
   for(int i=0;i<rates_total;i++)
     {
      LineBuffer[i]=close[i];
     }

//--- volveremos el valor prev_calculated para la siguiente llamada de la función
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de la línea trazada en el indicador         |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque de cambio del color de la línea
//--- obtenemos un número aleatorio
   int number=MathRand();
//--- el divisor del número es igual al tamaño del array colors[]
   int size=ArraySize(colors);
//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   int color_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- apuntaremos el color de la línea
   comm=comm+(string)colors[color_index];

//--- bloque de cambio del grosor de la línea
   number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5; // el grosor puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+", Width="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
```

```
//--- el divisor del número es igual al tamaño del array styles
   size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm=EnumToString(styles[style_index])+", "+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_SECTION

El estilo DRAW_SECTION dibuja los segmentos de color especificado a base de los valores del búfer indicador. El grosor, color y el estilo de la línea se puede establecer de la misma manera como para el estilo DRAW_LINE - con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite "vivificar" los indicadores, para que cambien su apariencia en función de la situación actual.

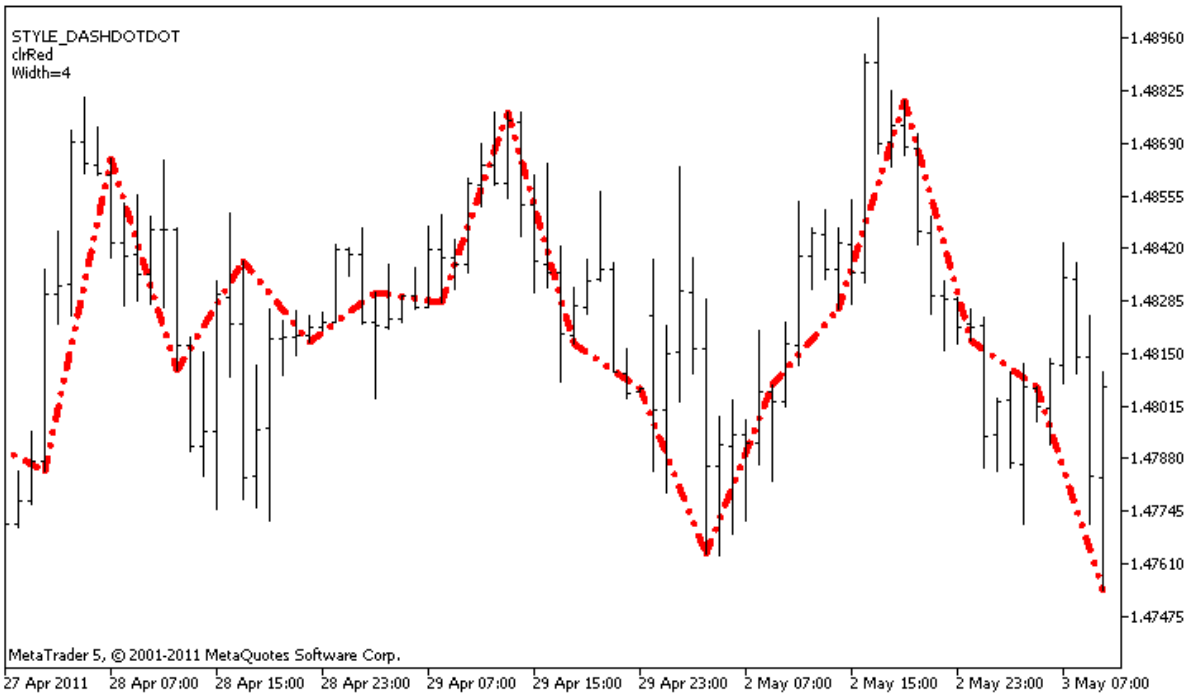
Los segmentos se trazan desde un valor no vacío hasta otro valor no vacío del búfer indicador, ignorando los valores vacíos. Para indicar qué valor se debe considerar "vacío", establezca este valor en la propiedad PLOT_EMPTY_VALUE. Por ejemplo, si un indicador debe dibujarse con segmentos sobre los valores no nulos, entonces hay que establecer el valor nulo como vacío:

```
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(índice_ de_construcción_DRAW_SECTION,PLOT_EMPTY_VALUE,0);
```

Rellene siempre todos los elementos del búfer indicador con valores de forma explícita, estableciendo el valor vacío para los elementos que no van a dibujarse.

El número de búfers requeridos para construir DRAW_SECTION — 1.

Aquí tenemos un ejemplo del indicador que traza segmentos entre los precios High y Low. El color, grosor y el estilo de **todos** los segmentos se cambian de forma aleatoria cada **N** tics.



Fíjense que inicialmente para la construcción gráfica **plot1** con el estilo DRAW_SECTION las propiedades se establecen mediante la directiva del compilador #property, y luego en la función OnCalculate() estas tres propiedades se establecen de forma aleatoria. El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                                DRAW_SECTION.mq5 |
```

```
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_SECTION"
#property description "Dibujamos con segmentos rectos dentro de cada bars barras"
#property description "El color, grosor y el estilo del segmento se cambia de forma a
#property description "dentro de cada N tics"

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Section
#property indicator_label1  "Section"
#property indicator_type1   DRAW_SECTION
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- parámetro input
input int      bars=5;           // longitud del segmento en barras
input int      N=5;              // número de tics para el cambio del estilo de segme
//--- búfer indicador para la construcción
double         SectionBuffer[];
//--- una variable auxiliar para calcular los extremos de segmentos
int            divider;
//--- array para almacenar colores
color colors[]={clrRed,clrBlue,clrGreen};
//--- array para almacenar estilos de trazado de la línea
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array con el búfer indicador
   SetIndexBuffer(0,SectionBuffer,INDICATOR_DATA);
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- comprobaremos el parámetro del indicador
   if(bars<=0)
     {
      PrintFormat("Valor del parámetro bar=%d inválido",bars);
      return(INIT_PARAMETERS_INCORRECT);
     }
   else divider=2*bars;
```

```
//---+
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- número de la barra a partir del cual empezaremos a calcular los valores del ind
   int start=0;
//--- si el indicador ha sido calculado antes, estableceremos start para la barra ant
   if(prev_calculated>0) start=prev_calculated-1;
//--- aquí están todos los cálculos de los valores del indicador
   for(int i=start;i<rates_total;i++)
     {
      //--- obtendremos el remanente de la división del número de la barra por 2*bars
      int rest=i%divider;
      //--- si el número de la barra se divide por 2*bars sin remanente
      if(rest==0)
        {
         //--- estableceremos el extremo del segmento en el precio High de esta barra
         SectionBuffer[i]=high[i];
        }
      //--- si la remanente de la división es igual a bars,
      else
        {
         //--- estableceremos el extremo del segmento en el precio High de esta barra
```

```
         if(rest==bars) SectionBuffer[i]=low[i];
         //--- si no encaja nada, omitimos esta barra - ponemos el valor 0
         else SectionBuffer[i]=0;
        }
      }
//--- volveremos el valor prev_calculated para la siguiente llamada de la función
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| cambia la apariencia del segmento en el indicador                |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque del cambio del color de la línea
   int number=MathRand(); // obtenemos un número aleatorio
//--- el divisor del número es igual al tamaño del array colors[]
   int size=ArraySize(colors);
//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   int color_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- apuntaremos el color de la línea
   comm=comm+"\r\n"+(string)colors[color_index];

//--- bloque de cambio del grosor de la línea
   number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5;   // el grosor puede ser de 0 a 4
//--- fijaremos el grosor
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
//--- el divisor del número es igual al tamaño del array styles
   size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el estilo de la línea
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

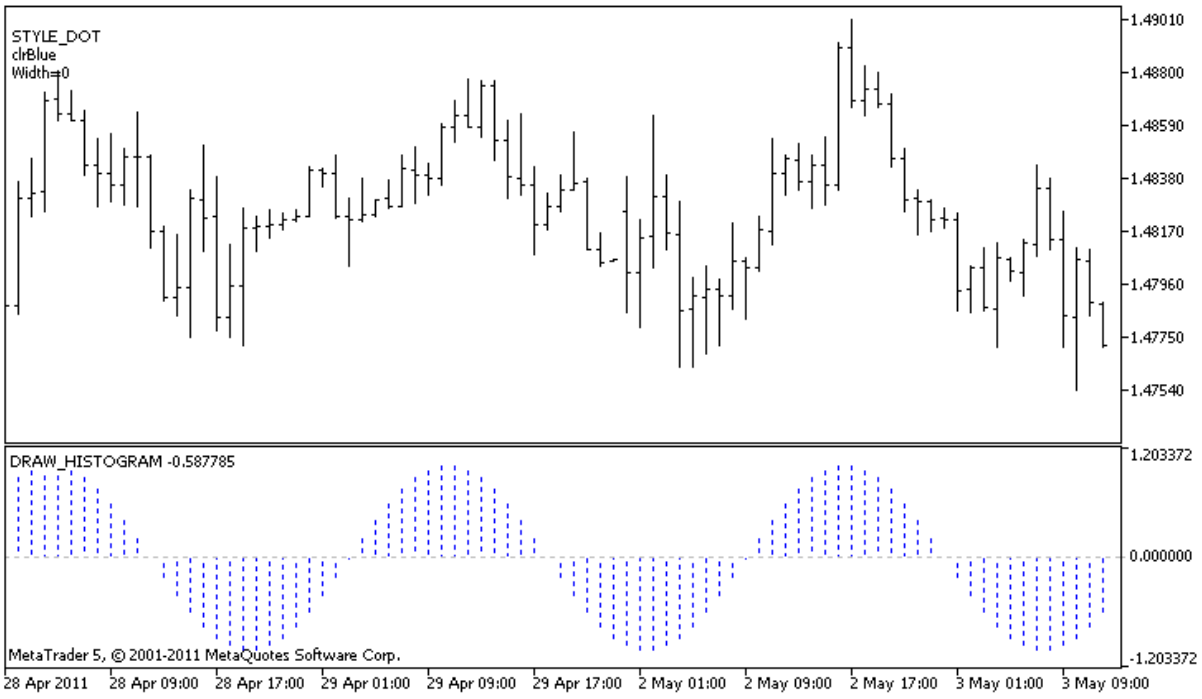
DRAW_HISTOGRAM

El estilo DRAW_HISTOGRAM dibuja un histograma como una secuencia de columnas de color especificado desde cero hasta el valor especificado . Los valores se cogen desde el búfer indicador. El grosor, color y el estilo de representación de la columna se puede establecer de la misma manera como para el estilo DRAW_LINE - con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite cambiar la apariencia del histograma en función de la situación actual.

Ya que en cada barra se dibuja una columna desde el nivel cero, es mejor utilizar DRAW_HISTOGRAM para mostrar en otra subventana del gráfico. Las más de las veces este tipo de construcción gráfica se utiliza para crear los indicadores del tipo oscilatorio, por ejemplo: Bears Power o OsMA. Para los valores vacíos que no se muestran, será suficiente indicarlos valores nulos.

El número de búfers requeridos para construir DRAW_HISTOGRAM — 1.

Aquí tenemos un ejemplo del indicador que dibuja una sinusoide de color especificado basándose en la función MathSin(). El color, grosor y el estilo de **todas** las columnas se cambian de forma aleatoria cada **N** tics. El parámetro bars determina el período de la sinusoide, eso quiere decir que dentro de una cantidad de barras especificada la sinusoide va a repetir su ciclo.



Fíjense que inicialmente para la construcción gráfica **plot1** con el estilo DRAW_HISTOGRAM las propiedades se establecen mediante la directiva del compilador #property, y luego en la función OnCalculate() estas tres propiedades se establecen de forma aleatoria. El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                                DRAW_HISTOGRAM.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
```

```
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_HISTOGRAM"
#property description "Dibuja la sinusoide como un histograma en otra ventana"
#property description "El color y el grosor de las columnas se cambian de forma aleat
#property description "dentro de cada N tics"
#property description "El parámetro bars establece el número de barras en el ciclo de

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Histogram
#property indicator_label1  "Histogram"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- parámetros input
input int      bars=30;          // período de la sinusoide en barras
input int      N=5;              // número de tics para el cambio del histograma
//--- indicator buffers
double         HistogramBuffer[];
//--- factor para obtener el ángulo 2Pi en radiánes al multiplicar por el parámetro b
double    multiplier;
//--- array para almacenar colores
color colors[]={clrRed,clrBlue,clrGreen};
//--- array para almacenar estilos de trazado de la línea
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,HistogramBuffer,INDICATOR_DATA);
//--- calcularemos el multiplicador
   if(bars>1)multiplier=2.*M_PI/bars;
   else
     {
      PrintFormat("Establezca el valor bars=%d mayor que 1",bars);
      //--- finalización anticipada del trabajo del indicador
      return(INIT_PARAMETERS_INCORRECT);
     }
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- cálculos de los valores del indicador
   int start=0;
//--- si el cálculo ya ha sido realizado en el arranque anterior de OnCalculate
   if(prev_calculated>0) start=prev_calculated-1; // fijaremos el inicio del cálculo
//--- llenamos el búfer indicador con valores
   for(int i=start;i<rates_total;i++)
     {
      HistogramBuffer[i]=sin(i*multiplier);
     }
//--- volveremos el valor prev_calculated para la siguiente llamada de la función
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de la línea en el indicador                 |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque de cambio del color de la línea
   int number=MathRand(); // obtenemos un número aleatorio
```

```
//--- el divisor del número es igual al tamaño del array colors[]
   int size=ArraySize(colors);
//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   int color_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- apuntaremos el color de la línea
   comm=comm+"\r\n"+(string)colors[color_index];

//--- bloque de cambio del grosor de la línea
   number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5;   // el grosor puede ser de 0 a 4
//--- fijaremos el grosor
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
//--- el divisor del número es igual al tamaño del array styles
   size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el estilo de la línea
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_HISTOGRAM2

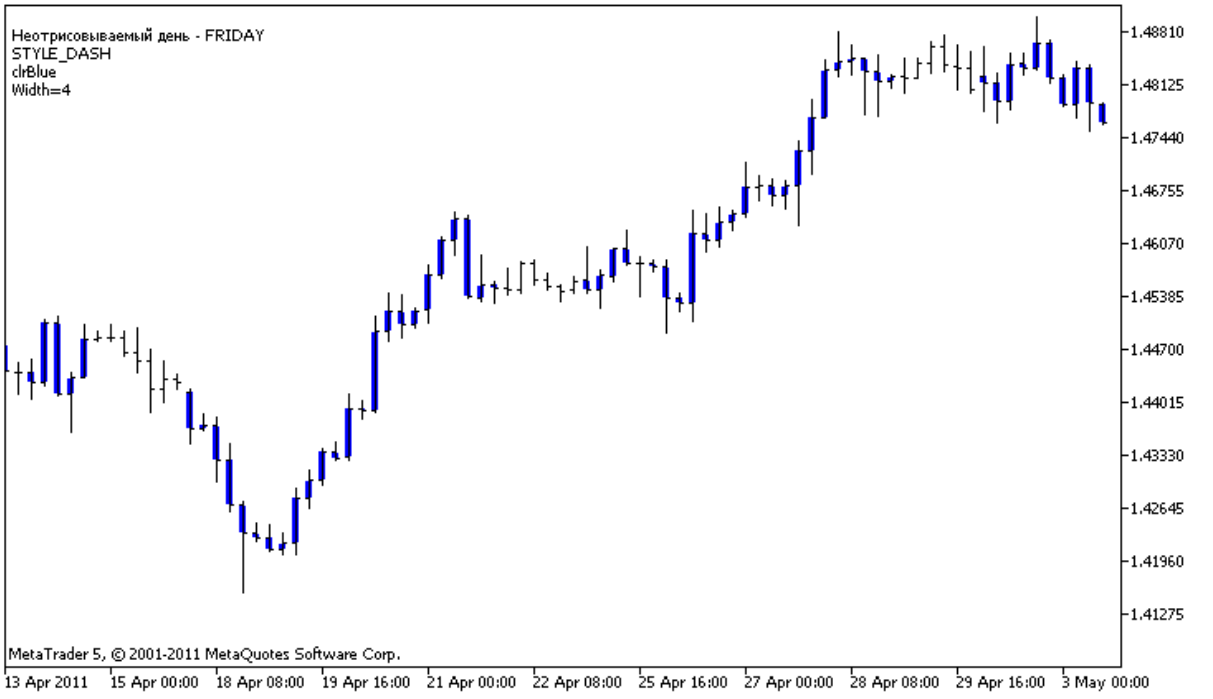
El estilo DRAW_HISTOGRAM2 dibuja un histograma de color especificado - segmentos verticales, usando valores de dos búfers indicadores. El grosor, color y el estilo de segmentos se puede establecer de la misma manera como para el estilo DRAW_LINE - con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite cambiar la apariencia del histograma en función de la situación actual.

El estilo DRAW_HISTOGRAM se puede utilizar tanto en una subventana separada del gráfico, como en la ventana principal. Los valores vacíos no se dibujan, todos los valores hay que establecer en los búfers indicadores de forma explícita. Los búfers no se inicializan con valores vacíos.

El número de búfers requeridos para construir DRAW_HISTOGRAM2 — 2.

Aquí tenemos un ejemplo del indicador que traza en cada barra un segmento vertical de color y grosor especificados entre los precios Open y Close. El color, grosor y el estilo de **todas** las columnas del histograma se cambian de forma aleatoria cada **N** tics. Cuando el indicador se inicia, en la función OnInit() se define al azar el número del día de la semana para el que el histograma no va a dibujarse - invisible_day. Para eso se establece el valor vacío PLOT_EMPTY_VALUE=0:

```
//--- estableceremos el valor vacío
   PlotIndexSetDouble(índice_ de_construcción_DRAW_SECTION,PLOT_EMPTY_VALUE,0);
```



Fíjense que inicialmente para la construcción gráfica **plot1** con el estilo DRAW_HISTOGRAM2 las propiedades se establecen mediante la directiva del compilador #property, y luego en la función OnCalculate() estas tres propiedades se establecen de forma aleatoria. El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                                DRAW_HISTOGRAM2.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
```

```
//|                                                 http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_HISTOGRAM2"
#property description "Dibuja en cada barra un segmento entre Open y Close"
#property description "El color, grosor y el estilo se cambian de forma aleatoria"
#property description "dentro de cada N tics"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot Histogram_2
#property indicator_label1  "Histogram_2"
#property indicator_type1   DRAW_HISTOGRAM2
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input int      N=5;              // número de tics para el cambio del histograma
//--- indicator buffers
double         Histogram_2Buffer1[];
double         Histogram_2Buffer2[];
//--- día de la semana para el que el indicador no se dibuja
int invisible_day;
//--- array para almacenar colores
color colors[]={clrRed,clrBlue,clrGreen};
//--- array para almacenar estilos de trazado de la línea
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Histogram_2Buffer1,INDICATOR_DATA);
   SetIndexBuffer(1,Histogram_2Buffer2,INDICATOR_DATA);
//--- estableceremos el valor vacío
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- obtenemos un número aleatorio de 0 a 5
   invisible_day=MathRand()%6;
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- cálculos de los valores del indicador
   int start=0;
//--- para obtener el día de la semana por la hora de apertura de cada barra
   MqlDateTime dt;
//--- si el cálculo ya ha sido realizado en el arranque anterior de OnCalculate
   if(prev_calculated>0) start=prev_calculated-1; // fijaremos el inicio del cálculo
//--- llenamos el búfer indicador con valores
   for(int i=start;i<rates_total;i++)
     {
      TimeToStruct(time[i],dt);
      if(dt.day_of_week==invisible_day)
        {
         Histogram_2Buffer1[i]=0;
         Histogram_2Buffer2[i]=0;
        }
      else
        {
         Histogram_2Buffer1[i]=open[i];
         Histogram_2Buffer2[i]=close[i];
        }
     }
//--- volveremos el valor prev_calculated para la siguiente llamada de la función
   return(rates_total);
  }
```

```
//+------------------------------------------------------------------+
//| cambia la apariencia de la línea en el indicador                 |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque del cambio del color de la línea
   int number=MathRand(); // obtenemos un número aleatorio
//--- el divisor del número es igual al tamaño del array colors[]
   int size=ArraySize(colors);
//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   int color_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- apuntaremos el color de la línea
   comm=comm+"\r\n"+(string)colors[color_index];

//--- bloque de cambio del grosor de la línea
   number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5;   // el grosor puede ser de 0 a 4
//--- fijaremos el grosor de la línea
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
//--- el divisor del número es igual al tamaño del array styles
   size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el estilo de la línea
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- agregaremos la información sobre el día que se ignora en los cálculos
   comm="\r\nDía no dibujable - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_ARROW

El estilo DRAW_ARROW dibuja en el gráfico las flechas de color especificado (símbolos del conjunto Wingdings) basándose en el valor del búfer indicador. El grosor y el color de los símbolo se puede establecer de la misma manera como para el estilo DRAW_LINE - con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite cambiar la apariencia del indicador en función de la situación actual.

El código del símbolo a mostrar en el gráfico se establece a través de la propiedad PLOT_ARROW.

```
//--- estableceremos el código del símbolo desde el fuente Wingdings para dibujar en
   PlotIndexSetInteger(0,PLOT_ARROW,code);
```

Por defecto, el valor de PLOT_ARROW=159 (un círculo).

Cada flecha prácticamente es un símbolo que tiene su alto y su punto de enlace, y puede cubrir alguna información importante en el gráfico (por ejemplo, el precio del cierre en la barra). Por eso se puede indicar adicionalmente el desplazamiento vertical en píxeles, que no depende de la escala del gráfico. Las flechas se desplazarán visualmente por la línea vertical a esta especificada cantidad de píxeles, aunque los valores del indicador se quedarán los mismos:

```
//--- estableceremos el desplazamiento de flechas por la línea vertical en píxeles
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
```

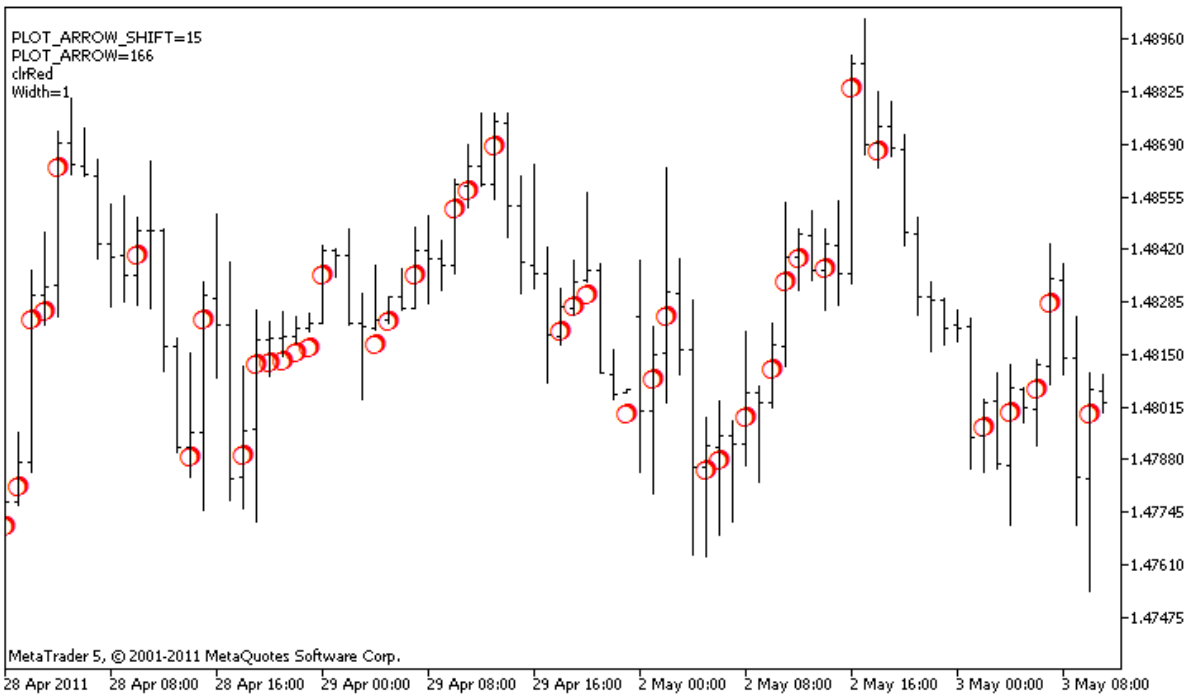
Un valor negativo de PLOT_ARROW_SHIFT significa el desplazamiento de las flechas hacia arriba, un valor positivo significa el desplazamiento de la flecha hacia abajo.

El estilo DRAW_ARROW se puede utilizar tanto en una subventana separada del gráfico, como en la ventana principal. Los valores vacíos no se dibujan y no se muestran en la "Ventana de datos", todos los valores hay que establecer en los búfers indicadores de forma explícita. Los búfers no se inicializan con valores vacíos.

```
//--- estableceremos el valor vacío
   PlotIndexSetDouble(índice_ de_construcción_DRAW_ARROW,PLOT_EMPTY_VALUE,0);
```

El número de búfers requeridos para construir DRAW_ARROW — 1.

Aquí tenemos un ejemplo del indicador que dibuja las flechas en cada barra cuyo precio de cierre Close es más alto que el precio de cierre de la barra anterior. El color, grosor, desplazamiento y el código del símbolo de **todas** las flechas se cambian de forma aleatoria cada **N** tics.



En el ejemplo, inicialmente para la construcción gráfica **plot1** con el estilo DRAW_ARROW las propiedades del color y el tamaño se establecen mediante la directiva del compilador #property, y luego en la función OnCalculate() las propiedades se establecen de forma aleatoria. El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                                   DRAW_ARROW.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_ARROW"
#property description "Dibuja las flechas determinadas por los caracteres de Unicode
#property description "El color, tamaño, desplazamiento y el código del símbolo de la
#property description "dentro de cada N tics"
#property description "El parámetro code establece el valor base: código=159 (círculo

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot Arrows
#property indicator_label1  "Arrows"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrGreen
#property indicator_width1  1
//--- parámetros input
```

```
input int      N=5;          // número de tics para el cambio
input ushort   code=159;     // código del símbolo a dibujar en DRAW_ARROW
//--- búfer indicador para la construcción
double         ArrowsBuffer[];
//--- array para almacenar colores
color colors[]={clrRed,clrBlue,clrGreen};
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ArrowsBuffer,INDICATOR_DATA);
//--- estableceremos el código del símbolo para dibujar en PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,code);
//--- estableceremos el desplazamiento de flechas por la línea vertical en píxeles
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
//--- estableceremos un 0 como valor vacío
   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[])
  {
   static int ticks=0;
//--- contamos los tics para el cambio del color, tamaño, desplazamiento y código de
   ticks++;
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }
```

```
//--- bloque para calcular los valores del indicador
   int start=1;
   if(prev_calculated>0) start=prev_calculated-1;
//--- ciclo del cálculo
   for(int i=1;i<rates_total;i++)
     {
      //--- si el precio actual Close es más alto que el anterior, colocamos la flech
      if(close[i]>close[i-1])
         ArrowsBuffer[i]=close[i];
      //--- en caso contrario, mostramos el valor cero
      else
         ArrowsBuffer[i]=0;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de símbolos en el indicador                 |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades del indicador
   string comm="";
//--- bloque del cambio del color de la flecha
   int number=MathRand(); // obtenemos un número aleatorio
//--- el divisor del número es igual al tamaño del array colors[]
   int size=ArraySize(colors);
//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   int color_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- apuntaremos el color de la línea
   comm=comm+"\r\n"+(string)colors[color_index];

//--- bloque del cambio del tamaño de flechas
   number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5;   // el tamaño puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el tamaño de flechas
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- bloque del cambio del código de la flecha (PLOT_ARROW)
   number=MathRand();
//--- obtendremos el remanente de la división de números enteros para calcular nuevo
   int code_add=number%20;
//--- estableceremos el nuevo código del símbolo como la suma code+code_add
```

```
   PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
//--- apuntaremos el código del símbolo PLOT_ARROW
   comm="\r\n"+"PLOT_ARROW="+IntegerToString(code+code_add)+comm;

//--- bloque del cambio del desplazamiento de flechas por la línea vertical en píxele
   number=MathRand();
//--- obtenemos el desplazamiento como el remanente de la división de números enteros
   int shift=20-number%41;
//--- estableceremos un nuevo desplazamiento de -20 a 20
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
//--- apuntaremos el desplazamiento PLOT_ARROW_SHIFT
   comm="\r\n"+"PLOT_ARROW_SHIFT="+IntegerToString(shift)+comm;

//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_ZIGZAG

El estilo DRAW_ZIGZAG dibuja segmentos de color especificado, usando valores de dos búfers indicadores. Este estilo es muy parecido al DRAW_SECTION, pero a diferencia del último, éste permite dibujar segmentos verticales dentro de los márgenes de una barra, si los valores de los dos búfers indicadores están establecidos para esta barra. Los segmentos se trazan desde un valor en el primer búfer indicador hasta un valor en el segundo. Ninguno de los dos búferes puede contener sólo valores vacíos. Si es así, no se dibuja nada.

El grosor, color y el estilo de la línea se puede establecer de la misma manera como para el estilo DRAW_SECTION - con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite "vivificar" los indicadores, para que cambien su apariencia en función de la situación actual.

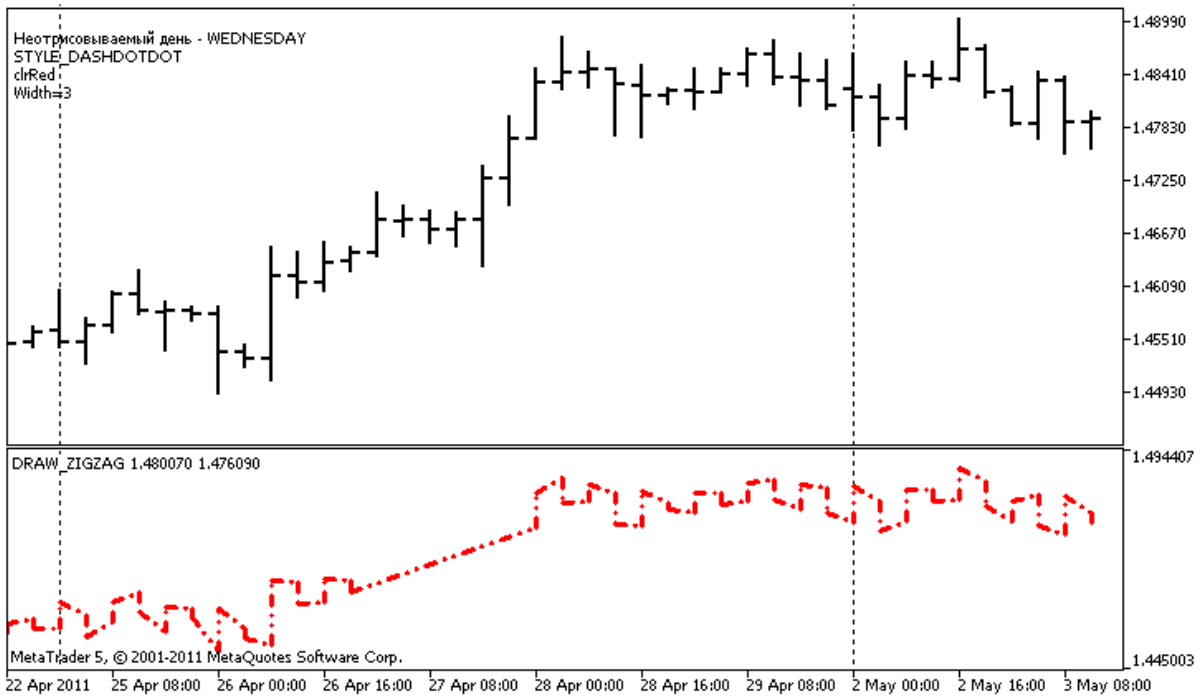
Los segmentos se trazan desde un valor no vacío de un búfer hasta otro valor no vacío de otro búfer indicador. Para indicar qué valor se debe considerar "vacío", establezca este valor en la propiedad PLOT_EMPTY_VALUE:

```
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(indice_ de_construcción_DRAW_ZIGZAG,PLOT_EMPTY_VALUE,0);
```

Rellene siempre los búfers indicadores con valores de forma explícita, para las barras que se ignoran indique en el búfer un valor vacío.

El número de búfers requeridos para construir DRAW_ZIGZAG — 2.

Aquí tenemos un ejemplo del indicador que traza la sierra a base de los precios High y Low. El color, grosor y el estilo de la línea del zigzag se cambian de forma aleatoria cada **N** tics.



Fíjense en que inicialmente para la construcción gráfica **plot1** con el estilo DRAW_ZIGZAG las propiedades se establecen mediante la directiva del compilador #property, y luego en la función OnCalculate() estas tres propiedades se establecen de forma aleatoria. El parámetro N está pasado a

los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                                  DRAW_ZIGZAG.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_ZIGZAG"
#property description "Dibuja con segmentos rectos \"la sierra\", saltando las barras
#property description "El día que se salta se elige de forma aleatoria al iniciar el
#property description "El color, grosor y el estilo de segmentos se cambia de forma a
#property description " dentro de cada N tics"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot ZigZag
#property indicator_label1  "ZigZag"
#property indicator_type1   DRAW_ZIGZAG
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- parámetros input
input int      N=5;              // número de tics a cambiar
//--- indicator buffers
double         ZigZagBuffer1[];
double         ZigZagBuffer2[];
//--- día de la semana para el que el indicador no se dibuja
int invisible_day;
//--- array para almacenar colores
color colors[]={clrRed,clrBlue,clrGreen};
//--- array para almacenar estilos de trazado de la línea
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de arrays con búfers indicadores
   SetIndexBuffer(0,ZigZagBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ZigZagBuffer2,INDICATOR_DATA);
//--- obtenemos un número aleatorio de 0 a 6, para este día el indicador no se dibuja
   invisible_day=MathRand()%6;
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
```

```
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetString(0,PLOT_LABEL,"ZigZag1;ZigZag2");
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número suficiente de tics
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- necesitaremos una estructura de tiempo para obtener el día de la semana de cada
   MqlDateTime dt;

//--- posición de inicio del cálculo
   int start=0;
//--- si el indicador se calcula en el tic anterior, empezamos el cálculo a partir de
   if(prev_calculated!=0) start=prev_calculated-1;
//--- ciclo de cálculos
   for(int i=start;i<rates_total;i++)
     {
      //--- apuntaremos en la estructura la hora de apertura de la barra
      TimeToStruct(time[i],dt);
      //--- si el día de la semana de esta barra es igual a invisible_day
      if(dt.day_of_week==invisible_day)
        {
         //--- apuntaremos en el búfer los valores vacíos para esta barra
```

```
         ZigZagBuffer1[i]=0;
         ZigZagBuffer2[i]=0;
        }
      //--- si el día de la semana es correcto, llenamos los búfers
      else
        {
         //--- si el número de la barra es par
         if(i%2==0)
           {
            //---  escribimos en el 1-r búfer High, en el 2-do Low
            ZigZagBuffer1[i]=high[i];
            ZigZagBuffer2[i]=low[i];
           }
         //--- número de la barra impar
         else
           {
            //--- llenamos la barra en sentido inverso
            ZigZagBuffer1[i]=low[i];
            ZigZagBuffer2[i]=high[i];
           }
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de segmentos en el zigzag                   |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades del ZigZag
   string comm="";
//--- bloque del cambio del color del zigzag
   int number=MathRand(); // obtenemos un número aleatorio
//--- el divisor del número es igual al tamaño del array colors[]
   int size=ArraySize(colors);
//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   int color_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- apuntaremos el color de la línea
   comm=comm+"\r\n"+(string)colors[color_index];

//--- bloque de cambio del grosor de la línea
   number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5;   // el grosor puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
```

```
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
//--- el divisor del número es igual al tamaño del array styles
   size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- agregaremos la información sobre el día que se ignora en los cálculos
   comm="\r\nDía no dibujable - "+EnumToString((ENUM_DAY_OF_WEEK)invisible_day)+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_FILLING

El estilo DRAW_FILLING dibuja un área de color entre los valores de dos búferes de indicadores. Prácticamente, este estilo traza dos líneas y colorea el espacio entre ellas en uno de dos colores predefinidos. Sirve para crear indicadores que dibujan canales. Ninguno de los dos búferes puede contener sólo valores vacíos. Si es así, no se dibuja nada.

Se puede definir dos colores para el relleno:

- el primer color se utiliza para las áreas donde los valores en el primer búfer indicador son más altos que los valores en el segundo búfer indicador;
- el segundo color se utiliza para las áreas donde los valores en el segundo búfer indicador son más altos que los valores en el primer búfer indicador.

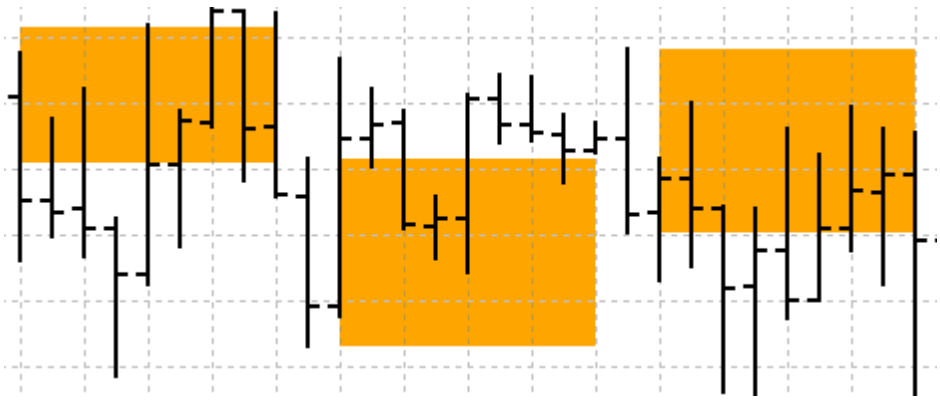
El color de relleno se puede definir con las directivas del compilador, o dinámicamente a través de la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite "vivificar" los indicadores, para que cambien su apariencia en función de la situación actual.

El indicador se calcula para todas las barras para las que los valores de ambos búferes de indicadores no son iguales a cero y no son iguales al valor vacío. Para indicar qué valor habrá que considerar como "vacío", especifíquelo en la propiedad PLOT_EMPTY_VALUE:

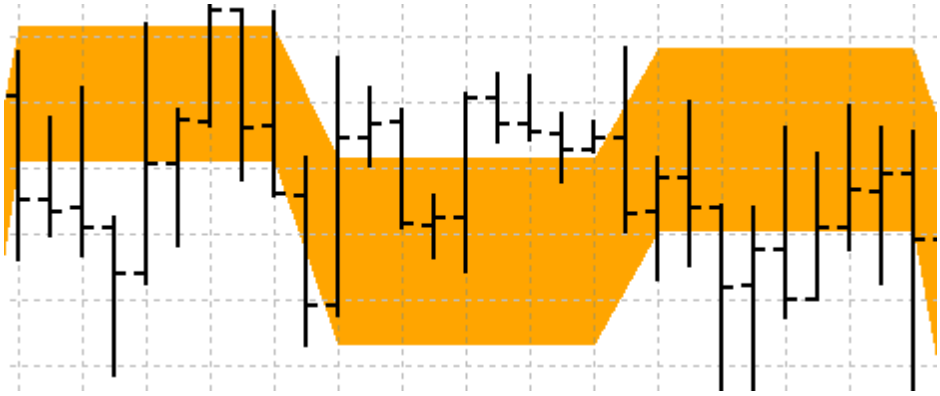
```
   #define INDICATOR_EMPTY_VALUE -1.0
   ...
//--- el valor INDICATOR_EMPTY_VALUE (valor vacío) no va a participar en el cálculo
   PlotIndexSetDouble(índice_de_construcción_DRAW_FILLING,PLOT_EMPTY_VALUE,INDICATOR_
```

El dibujo sobre las barras que no participan en el cálculo del indicador va a depender de los valores situados en los búferes de indicadores:

- Las barras para las que los valores de ambos búferes de indicadores son iguales a 0 no participan en el dibujo del indicador. Es decir el área con los valores iguales a cero no va a colorearse.



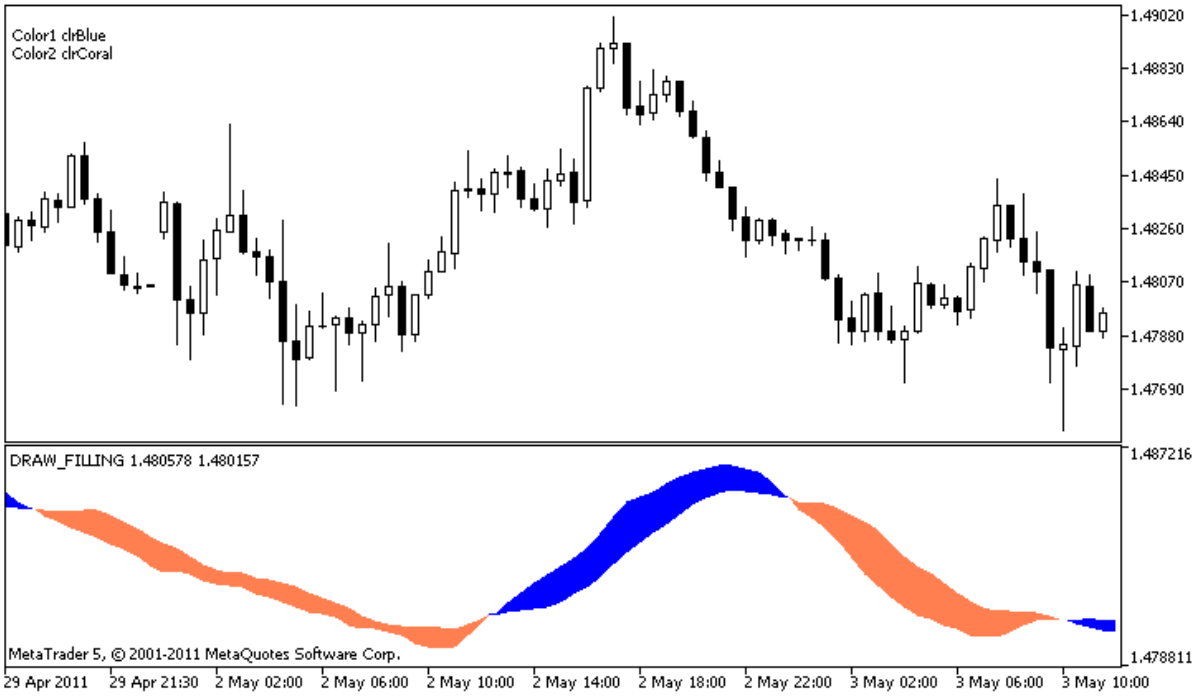
- Las barras para las que los valores de los búferes de indicadores son iguales a "valor vacío" participan en el dibujo del indicador. El área con valores vacíos va a colorearse de tal manera que las zonas con valores significativos se unan.



Es importante mencionar que si el "valor vacío" es igual a cero, las barras que no participan en el cálculo del indicador también van a colorearse.

El número de búfers requeridos para construir DRAW_FILLING — 2.

Aquí tenemos un ejemplo del indicador que dibuja en una ventana separada un canal entre dos medias móviles con diferentes períodos de promedio. El cambio de color durante el cruce de las medias muestra visualmente el cambio de la tendencia alcista y bajista. Los colores se cambian aleatoriamente cada **N** tics. El parámetro N está sacado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).



Fíjense en que inicialmente para la construcción gráfica **plot1** con el estilo DRAW_FILLING dos colores se establecen mediante la directiva del compilador #property, y luego en la función OnCalculate() nuevos colores se establecen de forma aleatoria.

```
//+------------------------------------------------------------------+
//|                                                 DRAW_FILLING.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
```

```
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_FILLING"
#property description "Dibuja en la ventana separada un canal entre dos medias móvile
#property description "El color del relleno del canal se cambia de forma aleatoria"
#property description "dentro de cada N tics"

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot Intersection
#property indicator_label1  "Intersection"
#property indicator_type1   DRAW_FILLING
#property indicator_color1  clrRed,clrBlue
#property indicator_width1  1
//--- parámetros input
input int      Fast=13;          // período de la media móvil rápida
input int      Slow=21;          // período de la media móvil lenta
input int      shift=1;          // desplazamiento de las medias móviles hacia el fut
input int      N=5;              // número de tics a cambiar
//--- búfers indicadores
double         IntersectionBuffer1[];
double         IntersectionBuffer2[];
int fast_handle;
int slow_handle;
//--- array para almacenar colores
color colors[]={clrRed,clrBlue,clrGreen,clrAquamarine,clrBlanchedAlmond,clrBrown,clrC
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,IntersectionBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,IntersectionBuffer2,INDICATOR_DATA);
//---
   PlotIndexSetInteger(0,PLOT_SHIFT,shift);
//---
   fast_handle=iMA(_Symbol,_Period,Fast,0,MODE_SMA,PRICE_CLOSE);
   slow_handle=iMA(_Symbol,_Period,Slow,0,MODE_SMA,PRICE_CLOSE);
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número suficiente de tics
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- hacemos el primer cálculo del indicador, o los datos han cambiado y se requiere
   if(prev_calculated==0)
     {
      //--- copiamos todos los valores de los indicadores a los búfers correspondient
      int copied1=CopyBuffer(fast_handle,0,0,rates_total,IntersectionBuffer1);
      int copied2=CopyBuffer(slow_handle,0,0,rates_total,IntersectionBuffer2);
     }
   else // llenamos sólo aquellos datos que se han actualizado
     {
      //--- obtendremos la diferencia en barras entre el arranque actual y el anterio
      int to_copy=rates_total-prev_calculated;
      //--- si no hay diferencia, igualmente copiaremos un valor - en la barra cero
      if(to_copy==0) to_copy=1;
      //--- copiamos to_copy valores al mismísimo final de los búfers indicadores
      int copied1=CopyBuffer(fast_handle,0,0,to_copy,IntersectionBuffer1);
      int copied2=CopyBuffer(slow_handle,0,0,to_copy,IntersectionBuffer2);
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Cambia los colores del relleno del canal                         |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
```

```
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque de cambio del color de la línea
   int number=MathRand(); // obtenemos un número aleatorio
//--- el divisor del número es igual al tamaño del array colors[]
   int size=ArraySize(colors);

//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   int color_index1=number%size;
//--- establecemos el primer color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,colors[color_index1]);
//--- apuntaremos el primer color
   comm=comm+"\r\nColor1 "+(string)colors[color_index1];

//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   number=MathRand(); // obtenemos un número aleatorio
   int color_index2=number%size;
//--- establecemos el segundo color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,colors[color_index2]);
//--- apuntaremos el segundo color
   comm=comm+"\r\nColor2 "+(string)colors[color_index2];
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_BARS

El estilo DRAW_BARS dibuja las barras basándose en los valores de cuatro búfers indicadores que contienen los precios Open, High, Low y Close. Se utiliza para crear sus propios indicadores personalizados en forma de barras, también en otra subventana del gráfico y para otros instrumentos financieros.

El color de las barras se puede definir con las directivas del compilador, o dinámicamente a través de la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite "vivificar" los indicadores, para que cambien su apariencia en función de la situación actual.

El indicador se dibuja sólo para las barras que tienen establecidos los valores no vacíos de **todos** los cuatro búfers indicadores. Para indicar qué valor se debe considerar "vacío", establezca este valor en la propiedad PLOT_EMPTY_VALUE:

```
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(índice_ de_construcción_DRAW_BARS,PLOT_EMPTY_VALUE,0);
```

Rellene siempre los búfers indicadores con valores de forma explícita, para las barras que se ignoran indique en el búfer un valor vacío.

El número de búfers requeridos para construir DRAW_BARS — 4. Todos los búfers que se utilizan para la construcción deben ir en serie uno detrás del otro en orden establecido: Open, High, Low y Close. Ninguno de los búfers puede contener sólo los valores vacíos, porque si es así, no se dibuja nada.

Aquí tenemos un ejemplo del indicador que dibuja en una ventana separada las barras para el instrumento financiero especificado. El color de las barras se cambia aleatoriamente cada **N** tics. El parámetro N está sacado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

Fíjense en que inicialmente para la construcción gráfica **plot1** con el estilo DRAW_BARS el color se establece mediante la directiva del compilador #property, y luego en la función OnCalculate() el color

se elige de forma aleatoria.

```
//+------------------------------------------------------------------+
//|                                                    DRAW_BARS.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_BARS"
#property description "Dibuja en la ventana separada las barras para el símbolo selec
#property description "El color y el grosor de las barras, igual que el estilo, se ca
#property description "cada N tics"

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   1
//--- plot Bars
#property indicator_label1  "Bars"
#property indicator_type1   DRAW_BARS
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- parámetros input
input int      N=5;              // número de tics para el cambio de apariencia
input int      bars=500;         // número de barras a mostrar
input bool     messages=false;   // mostrar mensajes en el log "Asesores Expertos"
//--- búfers indicadores
double         BarsBuffer1[];
double         BarsBuffer2[];
double         BarsBuffer3[];
double         BarsBuffer4[];
//--- nombre del símbolo
string symbol;
//--- array para almacenar colores
color colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- si bars es muy pequeño - finalizamos el trabajo antes de tiempo
   if(bars<50)
     {
      Comment("¡Por favor, indique el número más grande de barras! El trabajo del ind
      return(INIT_PARAMETERS_INCORRECT);
     }
```

```
//--- indicator buffers mapping
   SetIndexBuffer(0,BarsBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,BarsBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,BarsBuffer3,INDICATOR_DATA);
   SetIndexBuffer(3,BarsBuffer4,INDICATOR_DATA);
//--- nombre del símbolo para el que se dibujan las barras
   symbol=_Symbol;
//--- estableceremos la visualización del símbolo
   PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+syml
   IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS("+symbol+")");
//--- valor vacío
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0.0);
//---
   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[])
  {
   static int ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número suficiente de tics
   if(ticks>=N)
     {
      //--- escogeremos nuevo símbolo en la ventana "Observación del mercado"
      symbol=GetRandomSymbolName();
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();

      int tries=0;
      //--- haremos 5 intentos de llenar el búfer con los precios desde symbol
      while(!CopyFromSymbolToBuffers(symbol,rates_total) && tries<5)
        {
         //--- contador de llamadas a la función CopyFromSymbolToBuffers()
         tries++;
        }
      //--- actualizamos el contador de tics pasándolo a cero
```

```
      ticks=0;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//|  Llenamos los búfers indicadores con los precios                 |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,int total)
  {
//--- vamos a copiar los precios Open, High, Low y Close al array rates[]
   MqlRates rates[];
//--- contador de intentos
   int attempts=0;
//--- cantidad que se ha copiado ya
   int copied=0;
//--- hacemos 25 intentos de obtener la serie temporal para el símbolo necesario
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attem
     }
//--- si no se ha podido copiar la cantidad suficiente de barras
   if(copied!=bars)
     {
      //--- formaremos la cadena del mensaje
      string comm=StringFormat("Para el símbolo %s se ha logrado obtener sólo %d barr
                               name,
                               copied,
                               bars
                               );
      //--- mostraremos el mensaje en un comentario en la ventana principal del gráfi
      Comment(comm);
      //--- mostramos el mensaje
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- estableceremos la visualización del símbolo
      PlotIndexSetString(0,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;"+name+"
      IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_BARS("+name+")");
     }
//--- inicializaremos los búfers con valores vacíos
   ArrayInitialize(BarsBuffer1,0.0);
   ArrayInitialize(BarsBuffer2,0.0);
   ArrayInitialize(BarsBuffer3,0.0);
```

```
   ArrayInitialize(BarsBuffer4,0.0);
//--- copiamos los precios a los búfers
   for(int i=0;i<copied;i++)
     {
      //--- calcularemos el índice correspondiente para los búfers
      int buffer_index=total-copied+i;
      //--- escribimos los precios en los búfers
      BarsBuffer1[buffer_index]=rates[i].open;
      BarsBuffer2[buffer_index]=rates[i].high;
      BarsBuffer3[buffer_index]=rates[i].low;
      BarsBuffer4[buffer_index]=rates[i].close;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| devuelve aleatoriamente el símbolo desde Market Watch            |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- número de símbolos mostrados en la ventana "Observación del mercado"
   int symbols=SymbolsTotal(true);
//--- posición del símbolo en la lista - un número aleatorio de 0 a symbols
   int number=MathRand()%symbols;
//--- devolvemos el nombre del símbolo sobre la posición indicada
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de las barras                               |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de las barras
   string comm="";
//--- bloque del cambio del color de las barras
   int number=MathRand(); // obtenemos un número aleatorio
//--- el divisor del número es igual al tamaño del array colors[]
   int size=ArraySize(colors);
//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   int color_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- apuntaremos el color de la línea
   comm=comm+"\r\n"+(string)colors[color_index];

//--- bloque del cambio del grosor de las barras
   number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5;   // el grosor puede ser de 0 a 4
```

```
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- apuntamos el nombre del símbolo
   comm="\r\n"+symbol+comm;

//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_CANDLES

El estilo DRAW_CANDLES dibuja las velas japonesas basándose en los valores de cuatro búfers indicadores que contienen los precios Open, High, Low y Close. Se utiliza para crear sus propios indicadores personalizados en forma de velas japonesas, también en otra subventana del gráfico y para otros instrumentos financieros.

El color de las velas se puede definir usando las directivas del compilador, o dinámicamente a través de la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite "vivificar" los indicadores, para que cambien su apariencia en función de la situación actual.

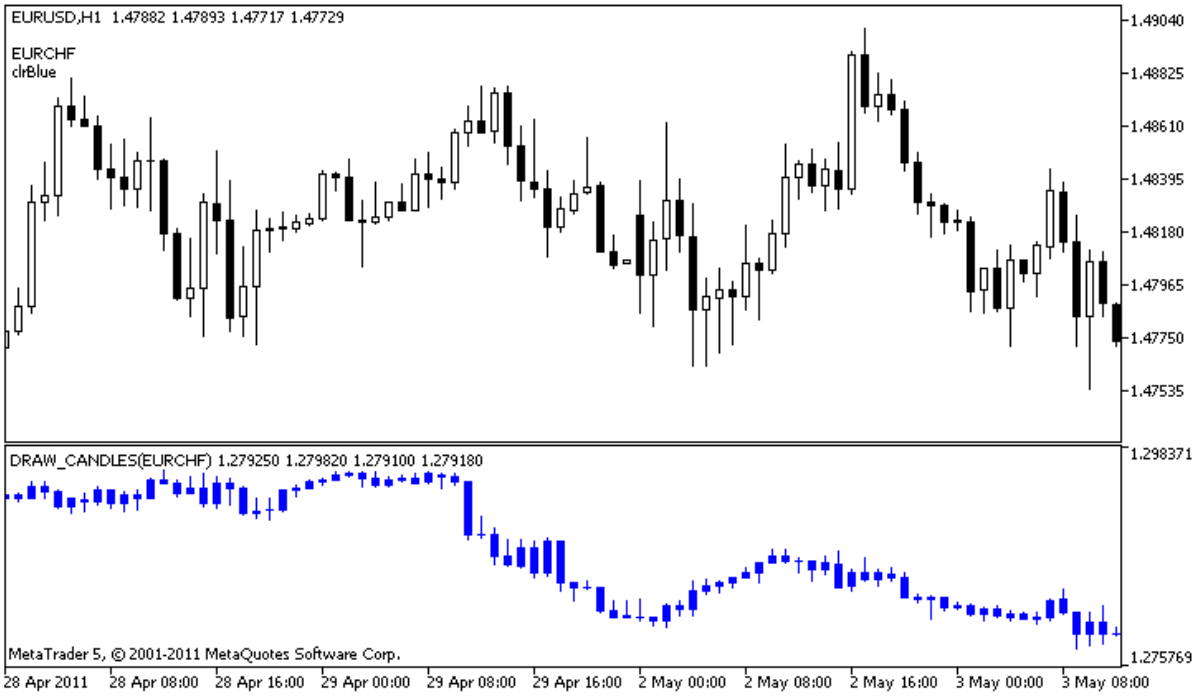
El indicador se dibuja sólo para las barras que tienen establecidos los valores no vacíos de **todos** los cuatro búfers indicadores. Para indicar qué valor se debe considerar "vacío", establezca este valor en la propiedad PLOT_EMPTY_VALUE:

```
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(índice_ de_construcción_DRAW_CANDLES,PLOT_EMPTY_VALUE,0);
```

Rellene siempre los búfers indicadores con valores de forma explícita, para las barras que se ignoran indique en el búfer un valor vacío.

El número de búfers requeridos para construir DRAW_CANDLES — 4. Todos los búfers que se utilizan para la construcción deben ir en serie uno detrás del otro en orden establecido: Open, High, Low y Close. Ninguno de los búfers puede contener sólo los valores vacíos, porque si es así, no se dibuja nada.

Aquí tenemos un ejemplo del indicador que dibuja en una ventana separada las velas japonesas para el instrumento financiero especificado. El color de las velas se cambia aleatoriamente cada **N** tics. El parámetro N está sacado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).



Fíjense en que inicialmente para la construcción gráfica **plot1** el color se establece mediante la

directiva del compilador #property, y luego en la funciónOnCalculate() se elige aleatoriamente un nuevo color desde la lista previamente preparada.

```
//+------------------------------------------------------------------+
//|                                                  DRAW_CANDLES.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_CANDLES."
#property description "Dibuja en la ventana separada las velas para el símbolo selecc
#property description " "
#property description "El color y el grosor de las velas, igual que el estilo, se cam
#property description "de forma aleatoria cada N tics."

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   1
//--- plot Bars
#property indicator_label1  "DRAW_CANDLES1"
#property indicator_type1   DRAW_CANDLES
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- parámetros input
input int      N=5;              // número de tics para el cambio de apariencia
input int      bars=500;         // número de barras a mostrar
input bool     messages=false;   // mostrar mensajes en el log "Asesores Expertos"
//--- búfers indicadores
double         Candle1Buffer1[];
double         Candle1Buffer2[];
double         Candle1Buffer3[];
double         Candle1Buffer4[];
//--- nombre del símbolo
string symbol;
//--- array para almacenar colores
color colors[]={clrRed,clrBlue,clrGreen,clrPurple,clrBrown,clrIndianRed};
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- si bars es muy pequeño - finalizamos el trabajo antes de tiempo
   if(bars<50)
     {
```

```
      Comment("¡Por favor, indique el número más grande de barras! El trabajo del ind
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- indicator buffers mapping
   SetIndexBuffer(0,Candle1Buffer1,INDICATOR_DATA);
   SetIndexBuffer(1,Candle1Buffer2,INDICATOR_DATA);
   SetIndexBuffer(2,Candle1Buffer3,INDICATOR_DATA);
   SetIndexBuffer(3,Candle1Buffer4,INDICATOR_DATA);
//--- valor vacío
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- nombre del símbolo para el que se dibujan las barras
   symbol=_Symbol;
//--- estableceremos la visualización del símbolo
   PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+syml
   IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_CANDLES("+symbol+")");
//---
   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 ticks=INT_MAX-100;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número suficiente de tics
   if(ticks>=N)
     {
      //--- escogeremos nuevo símbolo en la ventana "Observación del mercado"
      symbol=GetRandomSymbolName();
      //--- cambiamos la apariencia
      ChangeLineAppearance();
      //--- escogeremos nuevo símbolo en la ventana "Observación del mercado"
      int tries=0;
      //--- haremos 5 intentos de llenar el búfer plot1 con los precios desde symbol
      while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
            Candle1Buffer1,Candle1Buffer2,Candle1Buffer3,Candle1Buffer4)
            && tries<5)
```

```
         {
          //--- contador de llamadas a la función CopyFromSymbolToBuffers()
          tries++;
         }
       //--- actualizamos el contador de tics pasándolo a cero
       ticks=0;
      }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//|  Llena la vela indicada                                          |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,
                             int total,
                             int plot_index,
                             double &buff1[],
                             double &buff2[],
                             double &buff3[],
                             double &buff4[]
                             )
  {
//--- vamos a copiar los precios Open, High, Low y Close al array rates[]
   MqlRates rates[];
//--- contador de intentos
   int attempts=0;
//--- cantidad que se ha copiado ya
   int copied=0;
//--- hacemos 25 intentos de obtener la serie temporal para el símbolo necesario
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attem
     }
//--- si no se ha podido copiar la cantidad suficiente de barras
   if(copied!=bars)
     {
      //--- formaremos la cadena del mensaje
      string comm=StringFormat("Para el símbolo %s se ha logrado obtener sólo %d barr
                               name,
                               copied,
                               bars
                               );
      //--- mostraremos el mensaje en un comentario en la ventana principal del gráfi
      Comment(comm);
      //--- mostramos el mensaje
      if(messages) Print(comm);
```

```
      return(false);
     }
   else
     {
      //--- estableceremos la visualización del símbolo
      PlotIndexSetString(plot_index,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low
     }
//--- inicializaremos los búfers con valores vacíos
   ArrayInitialize(buff1,0.0);
   ArrayInitialize(buff2,0.0);
   ArrayInitialize(buff3,0.0);
   ArrayInitialize(buff4,0.0);
//--- en cada tic copiamos los precios a los búfers
   for(int i=0;i<copied;i++)
     {
      //--- calcularemos el índice correspondiente para los búfers
      int buffer_index=total-copied+i;
      //--- escribimos los precios en los búfers
      buff1[buffer_index]=rates[i].open;
      buff2[buffer_index]=rates[i].high;
      buff3[buffer_index]=rates[i].low;
      buff4[buffer_index]=rates[i].close;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| devuelve aleatoriamente el símbolo desde Market Watch            |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- número de símbolos mostrados en la ventana "Observación del mercado"
   int symbols=SymbolsTotal(true);
//--- posición del símbolo en la lista - un número aleatorio de 0 a symbols
   int number=MathRand()%symbols;
//--- devolvemos el nombre del símbolo sobre la posición indicada
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de las barras                               |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de las barras
   string comm="";
//--- bloque del cambio del color de las barras
   int number=MathRand(); // obtenemos un número aleatorio
//--- el divisor del número es igual al tamaño del array colors[]
   int size=ArraySize(colors);
```

```
//--- obtenemos el índice para seleccionar nuevo color como el remanente de la divisi
   int color_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_COLOR,colors[color_index]);
//--- apuntamos el color
   comm=comm+"\r\n"+(string)colors[color_index];
//--- apuntamos el nombre del símbolo
   comm="\r\n"+symbol+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_COLOR_LINE

El estilo DRAW_COLOR_LINE es la versión en color del estilo DRAW_LINE. Éste también traza una línea según los valores del búfer indicador. Pero este estilo, igual que todos los estilos de color en cuyo nombre figura **COLOR**, cuenta con un especial búfer indicador adicional que guarda el índice (número) del color desde un especial array de colores. De esta manera, se puede definir el color de cada sección de la línea indicando el índice del color que debe adquirir la línea en una barra en concreto.

Usted puede establecer el grosor, color y el estilo de la línea tanto con las directivas del compilador, como de forma dinámica, utilizando la función PlotIndexSetInteger(). La opción del cambio dinámico de las propiedades de la construcción gráfica permite crear indicadores "vivos", es decir, los que cambian su apariencia en función de la situación actual.

El número de búfers requeridos para construir DRAW_COLOR_LINE — 2:

- un búfer para almacenar los valores del indicador a base de los cuales se traza la línea;
- un búfer para almacenar el índice de color con el que se traza la línea en cada barra.

Se puede definir los colores con la directiva del compilador #property indicator_color1, separados por coma. El número de colores no puede superar 64.

```
//--- estableceremos 5 colores para colorear cada barra (se almacenan en un array esp
#property indicator_color1  clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (se pued
```

Aquí tenemos un ejemplo del indicador que traza una línea usando los precios de cierre de las barras Close. El grosor y el estilo de la línea se cambian de forma aleatoria cada N=5 tics.

Los colores para los segmentos de la línea también se cambian aleatoriamente en la función personalizada ChangeColors().

```
//+------------------------------------------------------------------+
//|  cambia el color de segmentos de la línea                        |
```

```
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- número de colores
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- definimos de forma aleatoria un color nuevo para cada índice de colores
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- obtendremos un número aleatorio
      int number=MathRand();
      //--- obtendremos un índice en el array col[] como el remanente de la división
      int i=number%size;
      //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
      PlotIndexSetInteger(0,                    //  número del estilo gráfico
                          PLOT_LINE_COLOR,      //  identificador de la propiedad
                          plot_color_ind,       //  índice del color donde escribirem
                          cols[i]);             //  nuevo color
      //--- apuntaremos los colores
      comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStrin
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
```

En el ejemplo se muestra una particularidad propia de las versiones "de colores" de los indicadores - para cambiar el color de un segmento de la línea no hace falta cambiar los valores en el búfer ColorLineColors[] (que almacena los índices de colores). Basta con definir nuevos colores en un array especial. Esto permite cambiar rápidamente el color para toda la construcción gráfica, modificando únicamente un pequeño array de colores a través de la función PlotIndexSetInteger().

Fíjense que inicialmente para la construcción gráfica **plot1** con el estilo DRAW_COLOR_LINE las propiedades se establecen mediante la directiva del compilador #property, y luego en la función OnCalculate() estas tres propiedades se establecen de forma aleatoria.

Los parámetros N y Length (longitud de los segmentos coloreados de la barra) están sacados a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                                DRAW_COLOR_LINE.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                                http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
```

```
#property description "Indicador para demostrar DRAW_COLOR_LINE"
#property description "Dibuja con segmentos de colores de 20 barras la línea a base d
#property description "El color, grosor y el estilo de segmentos de la línea se cambi
#property description "cada N tics"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot ColorLine
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
//--- estableceremos 5 colores para colorear cada barra (se almacenan en un array esp
#property indicator_color1  clrRed,clrBlue,clrGreen,clrOrange,clrDeepPink // (se pued
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- parámetros input
input int      N=5;           // número de tics a cambiar
input int      Length=20;     // longitud de cada sección de color en barras
int            line_colors=5; // número de colores definidos es igual a 5 - vea más a
//--- búfer para dibujar
double         ColorLineBuffer[];
//--- búfer para almacenar el color del trazado de la línea en cada barra
double         ColorLineColors[];

//--- el array para almacenar colores contiene 7 elementos
color colors[]={clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGold
//--- array para almacenar estilos de trazado de la línea
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array con el búfer indicador
   SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorLineColors,INDICATOR_COLOR_INDEX);
//--- inicialización del generador de números pseudoaleatorios
   MathSrand(GetTickCount());
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- cambiamos los colores con los que se dibujan los segmentos de colores de
      ChangeColors(colors,5);
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- bloque para calcular los valores del indicador
   for(int i=0;i<rates_total;i++)
     {
      //--- apuntamos el valor del indicador en el búfer
      ColorLineBuffer[i]=close[i];
      //--- ahora de forma aleatoria definimos un índice de color para esta barra
      int color_index=i%(5*Length);
      color_index=color_index/Length;
      //--- para esta barra la línea va a dibujarse con el color que se guarda bajo e
      ColorLineColors[i]=color_index;
     }

//--- volveremos el valor prev_calculated para la siguiente llamada de la función
   return(rates_total);
  }
//+------------------------------------------------------------------+
//|  cambia el color de segmentos de la línea                        |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- número de colores
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- definimos de forma aleatoria un color nuevo para cada índice de colores
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
```

```
      {
       //--- obtendremos un número aleatorio
       int number=MathRand();
       //--- obtendremos un índice en el array col[] como el remanente de la división
       int i=number%size;
       //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
       PlotIndexSetInteger(0,                    //  número del estilo gráfico
                           PLOT_LINE_COLOR,      //  identificador de la propiedad
                           plot_color_ind,       //  índice del color donde escribirem
                           cols[i]);             //  nuevo color
       //--- apuntaremos los colores
       comm=comm+StringFormat("LineColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStrin
       ChartSetString(0,CHART_COMMENT,comm);
      }
//---
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de la línea trazada en el indicador         |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque de cambio del grosor de la línea
   int number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5; // el grosor puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+" Width="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
//--- el divisor del número es igual al tamaño del array styles
   int size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm=EnumToString(styles[style_index])+", "+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_COLOR_SECTION

El estilo DRAW_COLOR_SECTION es la versión de colores del estilo DRAW_SECTION, pero a diferencia del último, permite colorear cada segmento con su propio color. El estilo DRAW_COLOR_SECTION, igual que todos los estilos de color en cuyo nombre figura **COLOR**, cuenta con un especial búfer indicador adicional que guarda el índice (número) del color desde un especial array de colores. De esta manera, se puede definir el color de cada segmento indicando un índice de color para la barra en la que cae el fin del segmento.

El grosor, color y el estilo de segmentos se puede establecer de la misma manera como para el estilo DRAW_SECTION - con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite "vivificar" los indicadores, para que cambien su apariencia en función de la situación actual.

Los segmentos se trazan desde un valor no vacío hasta otro valor no vacío del búfer indicador, ignorando los valores vacíos. Para indicar qué valor se debe considerar "vacío", establezca este valor en la propiedad PLOT_EMPTY_VALUE. Por ejemplo, si un indicador debe dibujarse con segmentos sobre los valores no nulos, entonces hay que establecer el valor nulo como vacío:

```
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(índice_ de_construcción_DRAW_COLOR_SECTION,PLOT_EMPTY_VALUE,0);
```

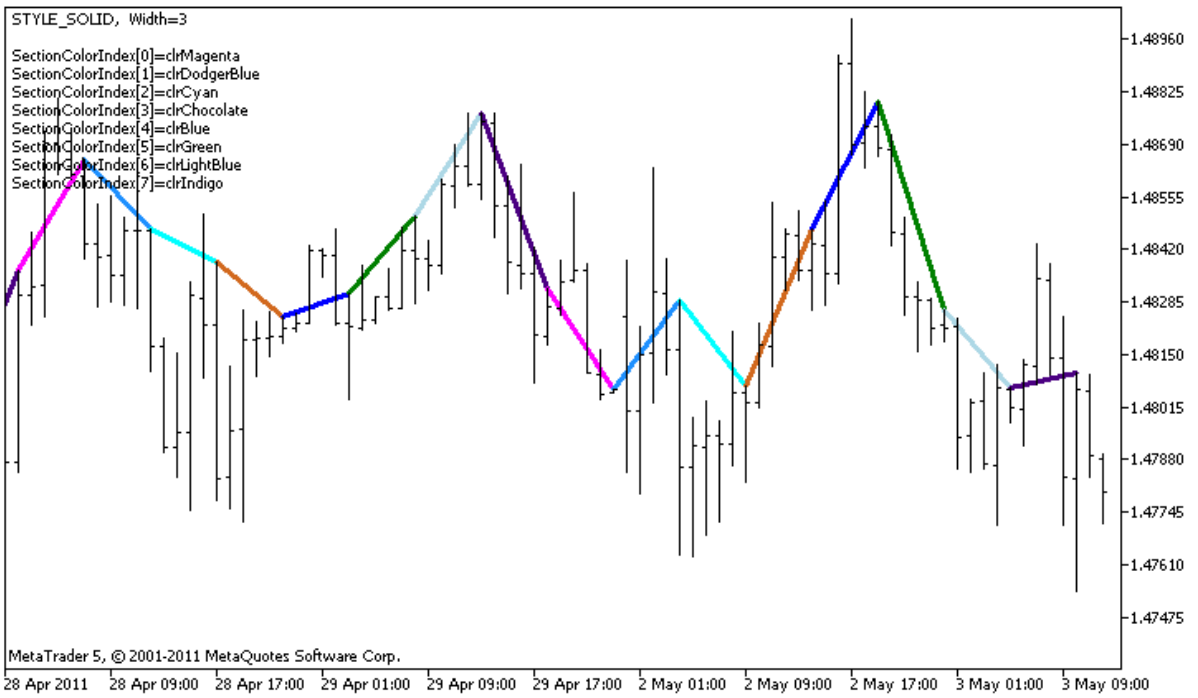
Rellene siempre todos los elementos del búfer indicador con valores de forma explícita, estableciendo el valor vacío para los elementos que no van a dibujarse.

El número de búfers requeridos para construir DRAW_COLOR_SECTION — 2:

- un búfer para almacenar los valores del indicador a base de los cuales se traza la línea;
- un búfer para almacenar el índice de color con el que se colorea el segmento (tiene sentido establecer sólo para los valores no vacíos).

Se puede definir los colores con la directiva del compilador #property indicator_color1, separados por coma. El número de colores no puede superar 64.

Aquí tenemos un ejemplo del indicador que dibuja segmentos de diferentes colores de 5 barras de longitud a base de los precios High. El color, grosor y el estilo de segmentos se cambian de forma aleatoria cada **N** tics.



Fíjense, inicialmente para la construcción gráfica **plot1** con el estilo DRAW_COLOR_SECTION se establecen 8 colores mediante la directiva del compilador #property. Luego en la función OnCalculate() los colores se establecen de forma aleatoria desde el array de colores colors[].

El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                          DRAW_COLOR_SECTION.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_COLOR_SECTION"
#property description "Dibuja con segmentos de colores de la longitud que corresponde
#property description "El color, grosor y el estilo del segmento se cambia de forma a
#property description "dentro de cada N tics"

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot ColorSection
#property indicator_label1  "ColorSection"
#property indicator_type1   DRAW_COLOR_SECTION
//--- estableceremos 8 colores para colorear los segmentos (se guardan en un array es
#property indicator_color1  clrRed,clrGold,clrMediumBlue,clrLime,clrMagenta,clrBrown,
#property indicator_style1  STYLE_SOLID
```

```
#property indicator_width1  1
//--- parámetros input
input int      N=5;                      // número de tics a cambiar
input int      bars_in_section=5;        // longitud de segmentos en barras
//--- una variable auxiliar para calcular los extremos de segmentos
int            divider;
int            color_sections;
//--- búfer para dibujar
double         ColorSectionBuffer[];
//--- búfer para almacenar el color del trazado de la línea en cada barra
double         ColorSectionColors[];
//--- el array para almacenar colores tiene 14 elementos
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPur
  };
//--- array para almacenar estilos de trazado de la línea
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorSectionBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorSectionColors,INDICATOR_COLOR_INDEX);
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- número de colores para colorear los segmentos
   color_sections=8;   //  ver comentario para la propiedad #property indicator_color
//--- comprobaremos el parámetro del indicador
   if(bars_in_section<=0)
     {
      PrintFormat("Segmento tiene una longitud inválida=%d",bars_in_section);
      return(INIT_PARAMETERS_INCORRECT);
     }
   else divider=color_sections*bars_in_section;
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- cambiamos colores con los que se dibujan segmentos
      ChangeColors(colors,color_sections);
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- número de la barra a partir del cual empezaremos a calcular los valores del ind
   int start=0;
//--- si el indicador ha sido calculado antes, estableceremos start para la barra ant
   if(prev_calculated>0) start=prev_calculated-1;
//--- aquí están todos los cálculos de los valores del indicador
   for(int i=start;i<rates_total;i++)
     {
      //--- si el número de la barra se divide sin remanente por la longitud_del_segm
      if(i%bars_in_section==0)
        {
         //--- estableceremos el extremo del segmento en el precio High de esta barra
         ColorSectionBuffer[i]=high[i];
         //--- remanente de la división del número de la barra por longitud_del_segme
         int rest=i%divider;
         //obtendremos el número del color =  de 0 a número_de_colores-1
         int color_indext=rest/bars_in_section;
         ColorSectionColors[i]=color_indext;
        }
      //--- si la remanente de la división es igual a bars,
      else
        {
         //--- si no encaja nada, omitimos esta barra - ponemos el valor 0
         ColorSectionBuffer[i]=0;
        }
     }
//--- volveremos el valor prev_calculated para la siguiente llamada de la función
   return(rates_total);
```

```
  }
//+------------------------------------------------------------------+
//|  cambia el color de segmentos de la línea                        |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- número de colores
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- definimos de forma aleatoria un color nuevo para cada índice de colores
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- obtendremos un número aleatorio
      int number=MathRand();
      //--- obtendremos un índice en el array col[] como el remanente de la división
      int i=number%size;
      //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
      PlotIndexSetInteger(0,                    //  número del estilo gráfico
                          PLOT_LINE_COLOR,      //  identificador de la propiedad
                          plot_color_ind,       //  índice del color donde escribirem
                          cols[i]);             //  nuevo color
      //--- apuntaremos los colores
      comm=comm+StringFormat("SectionColorIndex[%d]=%s \r\n",plot_color_ind,ColorToSt
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de la línea trazada en el indicador         |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque de cambio del grosor de la línea
   int number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5; // el grosor puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+" Width="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
//--- el divisor del número es igual al tamaño del array styles
```

```
   int size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm=EnumToString(styles[style_index])+", "+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_COLOR_HISTOGRAM

El estilo DRAW_COLOR_HISTOGRAM dibuja un histograma de columnas de diferentes colores desde cero hasta el valor especificado. Los valores se cogen desde el búfer indicador. Cada columna puede tener su propio color elegido de un conjunto previamente definido.

El grosor, color y el estilo del histograma se puede establecer de la misma manera como para el estilo DRAW_HISTOGRAM - con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite cambiar la apariencia del histograma en función de la situación actual.

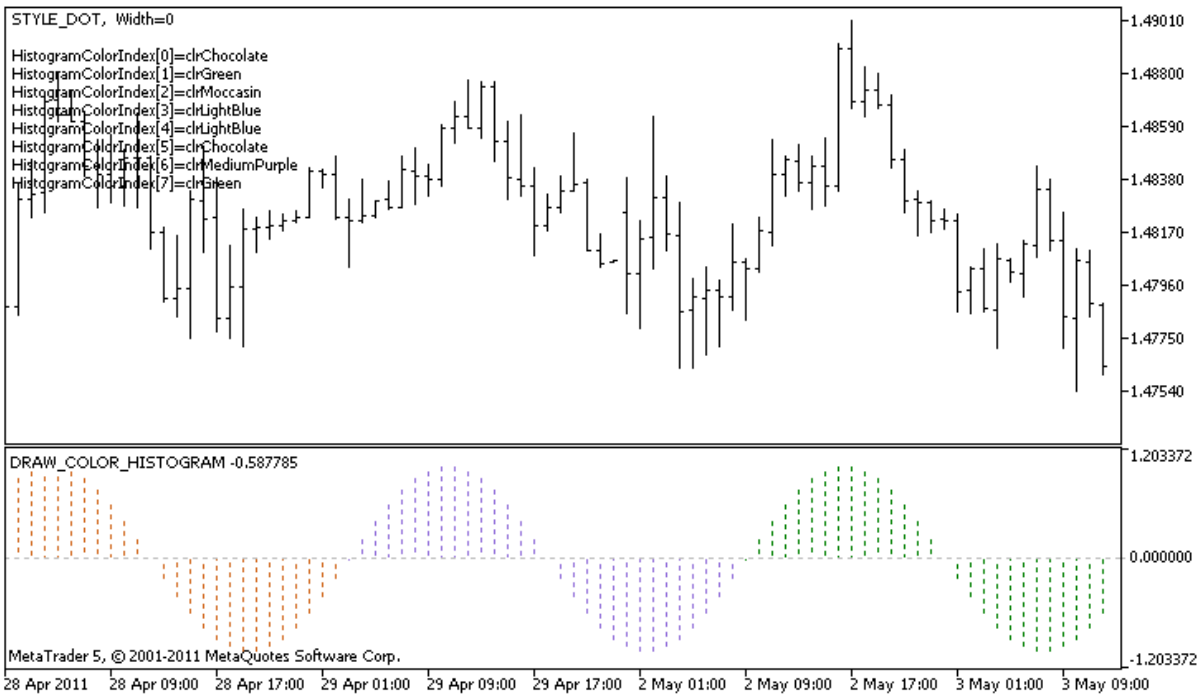
Ya que en cada barra se dibuja una columna desde el nivel cero, es mejor utilizar DRAW_COLOR_HISTOGRAM para mostrar en otra subventana del gráfico. Las más de las veces este tipo de construcción gráfica se utiliza para crear los indicadores del tipo oscilatorio, por ejemplo: Awesome Oscillator o Market Facilitation Index. Para los valores vacíos que no se muestran, será suficiente indicarlos valores nulos.

El número de búfers requeridos para construir DRAW_COLOR_HISTOGRAM — 2:

- un búfer para almacenar el valor no nulo del segmento vertical en cada barra, el segundo extremos del segmento siempre se encuentra en la línea cero del indicador;
- un búfer para almacenar el índice de color con el que se colorea el segmento (tiene sentido establecer sólo para los valores no vacíos).

Se puede definir los colores con la directiva del compilador #property indicator_color1, separados con coma. El número de colores no puede superar 64.

Aquí tenemos un ejemplo del indicador que dibuja una sinusoide de color especificado basándose en la función MathSin(). El color, grosor y el estilo de **todas** las columnas se cambian de forma aleatoria cada **N** tics. El parámetro bars determina el período de la sinusoide, eso quiere decir que dentro de una cantidad de barras especificada la sinusoide va a repetir su ciclo.



Fíjense, inicialmente para la construcción gráfica **plot1** con el estilo DRAW_COLOR_HISTOGRAM se establecen 5 colores mediante la directiva del compilador #property indicator_color1, y luego en la función OnCalculate() estos colores se eligen aleatoriamente de 14 colores que se guardan en el array colors[]. El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                          DRAW_COLOR_HISTOGRAM.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_COLOR_HISTOGRAM"
#property description "Dibuja la sinusoide como un histograma en otra ventana"
#property description "El color y el grosor de las columnas se cambian de forma aleat
#property description "dentro de cada N tics"
#property description "El parámetro bars establece el número de barras para la repeti

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- parámetros input
input int      bars=30;          // período de la sinusoide en barras
input int      N=5;              // número de tics para el cambio del histograma
//--- plot Color_Histogram
#property indicator_label1  "Color_Histogram"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
//--- estableceremos 8 colores para colorear los segmentos (se guardan en un array es
#property indicator_color1  clrRed,clrGreen,clrBlue,clrYellow,clrMagenta,clrCyan,clrM
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- búfer de valores
double         Color_HistogramBuffer[];
//--- búfer para los índices de colores
double         Color_HistogramColors[];
//--- factor para obtener el ángulo 2Pi en radiánes al multiplicar por el parámetro b
double         multiplier;
int            color_sections;
//--- el array para almacenar colores tiene 14 elementos
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPur
  };
//--- array para almacenar estilos de trazado de la línea
```

```
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Color_HistogramBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,Color_HistogramColors,INDICATOR_COLOR_INDEX);
//---- número de colores para colorear la sinusoide
   color_sections=8;   //  ver comentario para la propiedad #property indicator_color
//--- calcularemos el multiplicador
   if(bars>1)multiplier=2.*M_PI/bars;
   else
     {
      PrintFormat("Establezca el valor bars=%d mayor que 1",bars);
      //--- finalización anticipada del trabajo del indicador
      return(INIT_PARAMETERS_INCORRECT);
     }
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- cambiamos los colores con los que se dibuja el histograma
      ChangeColors(colors,color_sections);
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
```

```
     }

//--- cálculos de los valores del indicador
   int start=0;
//--- si el cálculo ya ha sido realizado en el arranque anterior de OnCalculate
   if(prev_calculated>0) start=prev_calculated-1; // fijaremos el inicio del cálculo
//--- llenamos el búfer indicador con valores
   for(int i=start;i<rates_total;i++)
     {
      //--- valor
      Color_HistogramBuffer[i]=sin(i*multiplier);
      //--- color
      int color_index=i%(bars*color_sections);
      color_index/=bars;
      Color_HistogramColors[i]=color_index;
     }
//--- volveremos el valor prev_calculated para la siguiente llamada de la función
   return(rates_total);
  }
  //+------------------------------------------------------------------+
//|  cambia el color de segmentos de la línea                          |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- número de colores
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- definimos de forma aleatoria un color nuevo para cada índice de colores
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- obtendremos un número aleatorio
      int number=MathRand();
      //--- obtendremos un índice en el array col[] como el remanente de la división
      int i=number%size;
      //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
      PlotIndexSetInteger(0,                    //  número del estilo gráfico
                          PLOT_LINE_COLOR,      //  identificador de la propiedad
                          plot_color_ind,       //  índice del color donde escribirem
                          cols[i]);             //  nuevo color
      //--- apuntaremos los colores
      comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorTo
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
```

```
//| cambia la apariencia de la línea trazada en el indicador            |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque de cambio del grosor de la línea
   int number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5; // el grosor puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+" Width="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
//--- el divisor del número es igual al tamaño del array styles
   int size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm=EnumToString(styles[style_index])+", "+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_COLOR_HISTOGRAM2

El estilo DRAW_COLOR_HISTOGRAM2 dibuja un histograma de color especificado – segmentos verticales, usando valores de dos búfers indicadores. Pero a diferencia del estilo DRAW_HISTOGRAM2 unicolor, en este estilo cada columna del histograma puede obtener su propio color desde el conjunto predefinido. Los valores de los extremos de segmentos se cogen del búfer indicador.

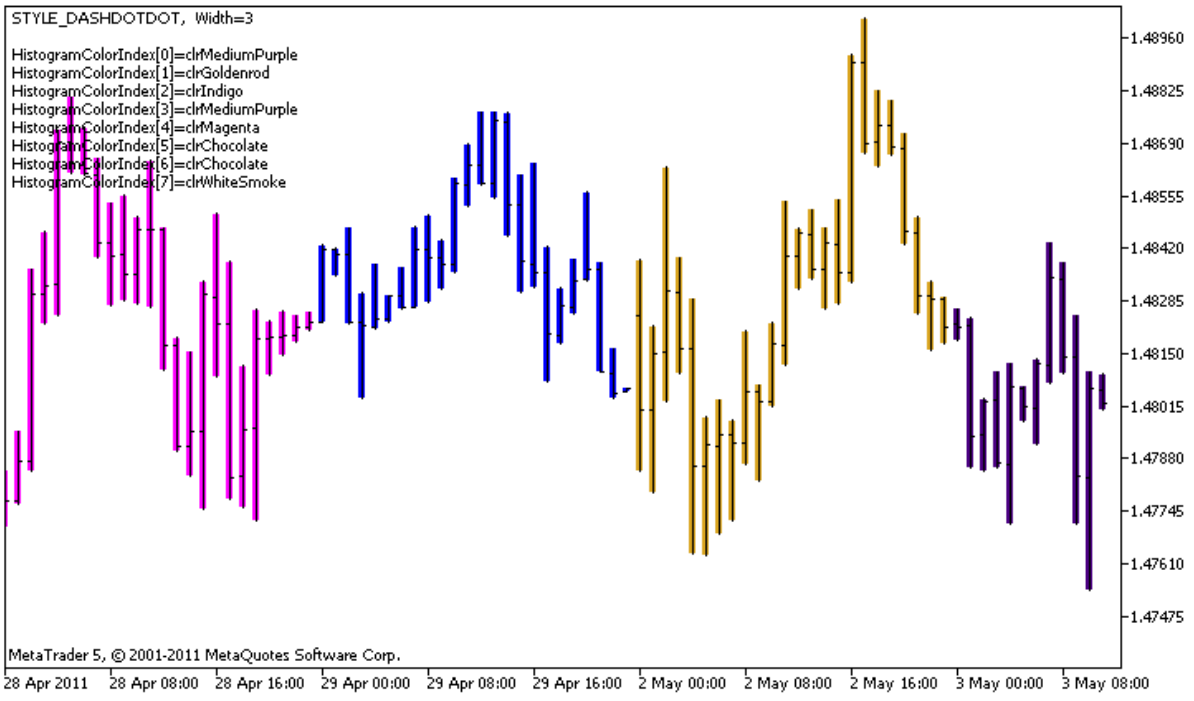
El grosor, colores y el estilo del histograma se puede establecer de la misma manera como para el estilo DRAW_HISTOGRAM2 – con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite cambiar la apariencia del histograma en función de la situación actual.

El estilo DRAW_COLOR_HISTOGRAM2 se puede utilizar tanto en una subventana separada del gráfico, como en la ventana principal. Los valores vacíos no se dibujan, todos los valores hay que establecer en los búfers indicadores de forma explícita. Los búfers no se inicializan con valores vacíos.

El número de búfers requeridos para construir DRAW_COLOR_HISTOGRAM2 — 3:

- dos búfers para guardar el extremo superior e inferior del segmento vertical en cada barra;
- un búfer para almacenar el índice de color con el que se colorea el segmento (tiene sentido establecer sólo para los valores no vacíos).

Aquí tenemos un ejemplo del indicador que dibuja con un color especificado un histograma entre los precios High y Low. Para cada día de la semana las líneas del histograma tendrán su color. El color de cada día, grosor y el estilo del histograma se cambian de forma aleatoria cada **N** tics.



Fíjense, inicialmente para la construcción gráfica **plot1** con el estilo DRAW_COLOR_HISTOGRAM2 se establecen 5 colores mediante la directiva del compilador #property indicator_color1, y luego en la función OnCalculate() estos colores se eligen aleatoriamente de 14 colores que se guardan en el array colors[].

El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de

establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                          DRAW_COLOR_HISTOGRAM2.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_COLOR_HISTOGRAM2"
#property description "Dibuja en cada barra un segmento entre Open y Close"
#property description "El color, grosor y el estilo se cambian de forma aleatoria"
#property description "dentro de cada N tics"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   1
//--- plot ColorHistogram_2
#property indicator_label1  "ColorHistogram_2"
#property indicator_type1   DRAW_COLOR_HISTOGRAM2
//--- estableceremos 5 colores para colorear el histograma por días de la semana (se
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- parámetro input
input int      N=5;              // número de tics para el cambio del histograma
int            color_sections;
//--- búfers de valores
double         ColorHistogram_2Buffer1[];
double         ColorHistogram_2Buffer2[];
//--- búfer para los índices del color
double         ColorHistogram_2Colors[];
//--- el array para almacenar colores tiene 14 elementos
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPur
  };
//--- array para almacenar estilos de trazado de la línea
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
```

```
   SetIndexBuffer(0,ColorHistogram_2Buffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ColorHistogram_2Buffer2,INDICATOR_DATA);
   SetIndexBuffer(2,ColorHistogram_2Colors,INDICATOR_COLOR_INDEX);
//--- estableceremos el valor vacío
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- número de colores para colorear la sinusoide
   color_sections=8;   //  ver comentario para la propiedad #property indicator_color
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- cambiamos los colores con los que se dibuja el histograma
      ChangeColors(colors,color_sections);
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- cálculos de los valores del indicador
   int start=0;
//--- para obtener el día de la semana por la hora de apertura de cada barra
   MqlDateTime dt;
//--- si el cálculo ya ha sido realizado en el arranque anterior de OnCalculate
   if(prev_calculated>0) start=prev_calculated-1; // fijaremos el inicio del cálculo
//--- llenamos el búfer indicador con valores
   for(int i=start;i<rates_total;i++)
     {
      TimeToStruct(time[i],dt);
```

```
      //--- valor
      ColorHistogram_2Buffer1[i]=high[i];
      ColorHistogram_2Buffer2[i]=low[i];
      //--- establecemos el índice de color según el día de la semana
      int day=dt.day_of_week;
      ColorHistogram_2Colors[i]=day;
     }
//--- volveremos el valor prev_calculated para la siguiente llamada de la función
   return(rates_total);
  }
//+------------------------------------------------------------------+
//|  cambia el color de segmentos de la línea                        |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- número de colores
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- definimos de forma aleatoria un color nuevo para cada índice de colores
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- obtendremos un número aleatorio
      int number=MathRand();
      //--- obtendremos un índice en el array col[] como el remanente de la división
      int i=number%size;
      //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
      PlotIndexSetInteger(0,                    //  número del estilo gráfico
                          PLOT_LINE_COLOR,      //  identificador de la propiedad
                          plot_color_ind,       //  índice del color donde escribirem
                          cols[i]);             //  nuevo color
      //--- apuntaremos los colores
      comm=comm+StringFormat("HistogramColorIndex[%d]=%s \r\n",plot_color_ind,ColorTo
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de la línea trazada en el indicador         |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque de cambio del grosor de la línea
   int number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
```

```
   int width=number%5; // el grosor puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+" Width="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
//--- el divisor del número es igual al tamaño del array styles
   int size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm=EnumToString(styles[style_index])+", "+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_COLOR_ARROW

El estilo DRAW_COLOR_ARROW dibuja en el gráfico las flechas de color (símbolos del conjunto Wingdings) basándose en el valor del búfer indicador. A diferencia del estilo DRAW_ARROW, este estilo permite establecer para cada símbolo su color desde un conjunto predefinido de colores especificados por medio de la propiedad indicator_color1.

El grosor y el color de los símbolo se puede establecer de la misma manera como para el estilo DRAW_ARROW - con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite cambiar la apariencia del indicador en función de la situación actual.

El código del símbolo a mostrar en el gráfico se establece a través de la propiedad PLOT_ARROW.

```
//--- estableceremos el código del símbolo desde el fuente Wingdings para dibujar en
   PlotIndexSetInteger(0,PLOT_ARROW,code);
```

Por defecto, el valor de PLOT_ARROW=159 (un círculo).

Cada flecha prácticamente es un símbolo que tiene su alto y su punto de enlace, y puede cubrir alguna información importante en el gráfico (por ejemplo, el precio del cierre en la barra). Por eso se puede indicar adicionalmente el desplazamiento vertical en píxeles, que no depende de la escala del gráfico. Las flechas se desplazarán visualmente por la línea vertical a esta especificada cantidad de píxeles, aunque los valores del indicador se quedarán los mismos:

```
//--- estableceremos el desplazamiento de flechas por la línea vertical en píxeles
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
```

Un valor negativo de PLOT_ARROW_SHIFT significa el desplazamiento de las flechas hacia arriba, un valor positivo significa el desplazamiento de la flecha hacia abajo.

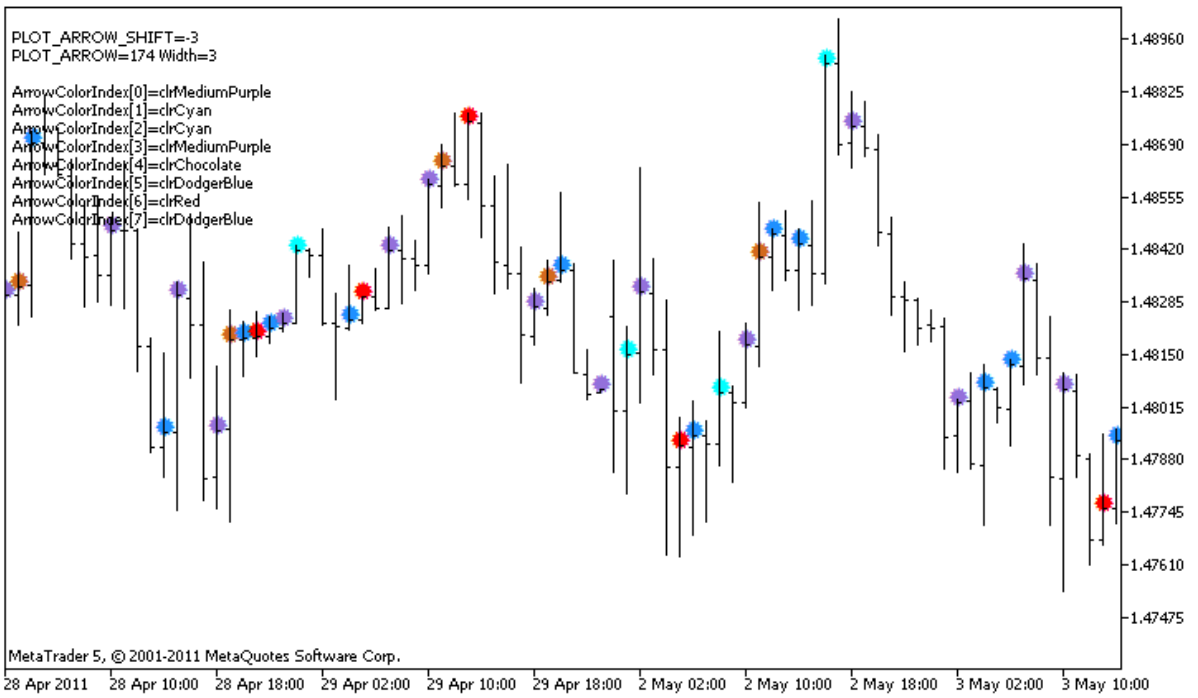
El estilo DRAW_COLOR_ARROW se puede utilizar tanto en una subventana separada del gráfico, como en la ventana principal. Los valores vacíos no se dibujan y no se muestran en la "Ventana de datos", todos los valores hay que establecer en los búfers indicadores de forma explícita. Los búfers no se inicializan con valores vacíos.

```
//--- estableceremos el valor vacío
   PlotIndexSetDouble(índice_ de_construcción_DRAW_COLOR_ARROW,PLOT_EMPTY_VALUE,0);
```

El número de búfers requeridos para construir DRAW_COLOR_ARROW — 2:

- un búfer para almacenar los valores del precio a base del cual se dibuja el símbolo (más el desplazamiento en píxeles que se fila por la propiedad PLOT_ARROW_SHIFT);
- un búfer para almacenar el índice de color con el que se colorea la flecha (tiene sentido establecer sólo para los valores no vacíos).

Aquí tenemos un ejemplo del indicador que dibuja las flechas en cada barra cuyo precio de cierre Close es más alto que el precio de cierre de la barra anterior. El grosor, desplazamiento y el código del símbolo de **todas** las flechas se cambian de forma aleatoria cada **N** tics. El precio del símbolo depende del número de la barra en la que está dibujado.



En el ejemplo, inicialmente para la construcción gráfica **plot1** con el estilo DRAW_COLOR_ARROW las propiedades del color y el tamaño se establecen mediante la directiva del compilador #property, y luego en la función OnCalculate() las propiedades se cambian de forma aleatoria. El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

Fíjense, inicialmente se establecen 8 colores mediante la directiva del compilador #property, y luego en la función OnCalculate() el color se elige aleatoriamente de 14 colores que se guardan en el array colors[].

```
//+------------------------------------------------------------------+
//|                                                DRAW_COLOR_ARROW.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_COLOR_ARROW"
#property description "Dibuja en el gráfico las flechas de diferentes colores determi
#property description "El color, tamaño, desplazamiento y el código del símbolo de la
#property description " de forma aleatoria cada N tics"
#property description "El parámetro code establece el valor base: código=159 (círculo

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//--- plot ColorArrow
#property indicator_label1  "ColorArrow"
```

```
#property indicator_type1   DRAW_COLOR_ARROW
//--- estableceremos 8 colores para colorear el histograma por días de la semana (se
#property indicator_color1  clrRed,clrBlue,clrSeaGreen,clrGold,clrDarkOrange,clrMagen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1

//--- parámetros input
input int       N=5;          // número de tics para el cambio
input ushort    code=159;     // código del símbolo a dibujar en DRAW_ARROW
int             color_sections;
//--- búfer indicador para la construcción
double          ColorArrowBuffer[];
//--- búfer para guardar los índices del color
double          ColorArrowColors[];
//--- el array para almacenar colores tiene 14 elementos
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPur
  };
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorArrowBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorArrowColors,INDICATOR_COLOR_INDEX);
//--- estableceremos el código del símbolo para dibujar en PLOT_ARROW
   PlotIndexSetInteger(0,PLOT_ARROW,code);
//--- estableceremos el desplazamiento de flechas por la línea vertical en píxeles
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,5);
//--- estableceremos un 0 como valor vacío
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//---- número de colores para colorear la sinusoide
   color_sections=8;   //  ver comentario para la propiedad #property indicator_color
//---
   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 ticks=0;
//--- contamos los tics para el cambio del color, tamaño, desplazamiento y código de
   ticks++;
//--- si tenemos acumulado un número crítico de tics,
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de las flechas
      ChangeLineAppearance();
      //--- cambiamos los colores con los que se dibuja el histograma
      ChangeColors(colors,color_sections);
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- bloque para calcular los valores del indicador
   int start=1;
   if(prev_calculated>0) start=prev_calculated-1;
//--- ciclo del cálculo
   for(int i=1;i<rates_total;i++)
     {
      //--- si el precio actual Close es más alto que el anterior, colocamos la flech
      if(close[i]>close[i-1])
         ColorArrowBuffer[i]=close[i];
      //--- en caso contrario, mostramos el valor cero
      else
         ColorArrowBuffer[i]=0;
      //--- color de la flecha
      int index=i%color_sections;
      ColorArrowColors[i]=index;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//|  cambia el color de segmentos de la línea                        |
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- número de colores
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";
```

```
//--- definimos de forma aleatoria un color nuevo para cada índice de colores
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- obtendremos un número aleatorio
      int number=MathRand();
      //--- obtendremos un índice en el array col[] como el remanente de la división
      int i=number%size;
      //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
      PlotIndexSetInteger(0,                    //  número del estilo gráfico
                          PLOT_LINE_COLOR,      //  identificador de la propiedad
                          plot_color_ind,       //  índice del color donde escribirem
                          cols[i]);             //  nuevo color
      //--- apuntaremos los colores
      comm=comm+StringFormat("ArrowColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStri
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de la línea trazada en el indicador         |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de la línea
   string comm="";
//--- bloque de cambio del grosor de la línea
   int number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5; // el grosor puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+" Width="+IntegerToString(width);

//--- bloque del cambio del código de la flecha (PLOT_ARROW)
   number=MathRand();
//--- obtendremos el remanente de la división de números enteros para calcular nuevo
   int code_add=number%20;
//--- estableceremos el nuevo código del símbolo como la suma code+code_add
   PlotIndexSetInteger(0,PLOT_ARROW,code+code_add);
//--- apuntaremos el código del símbolo PLOT_ARROW
   comm="\r\n"+"PLOT_ARROW="+IntegerToString(code+code_add)+comm;

//--- bloque del cambio del desplazamiento de flechas por la línea vertical en píxele
   number=MathRand();
//--- obtenemos el desplazamiento como el remanente de la división de números enteros
   int shift=20-number%41;
//--- estableceremos nuevo desplazamiento
```

```
   PlotIndexSetInteger(0,PLOT_ARROW_SHIFT,shift);
//--- apuntaremos el desplazamiento PLOT_ARROW_SHIFT
   comm="\r\n"+"PLOT_ARROW_SHIFT="+IntegerToString(shift)+comm;

//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_COLOR_ZIGZAG

El estilo DRAW_COLOR_ZIGZAG dibuja segmentos de diferentes colores, usando valores de dos búfers indicadores. Este estilo es la versión de colores del estilo DRAW_ZIGZAG. Es decir, permite fijar para cada segmento su propio color desde un conjunto de colores predefinido previamente. Los segmentos se trazan desde un valor en el primer búfer indicador hasta un valor en el segundo. Ninguno de los dos búferes puede contener sólo valores vacíos. Si es así, no se dibuja nada.

El grosor, color y el estilo de la línea se puede establecer de la misma manera como para el estilo DRAW_ZIGZAG - con las directivas del compilador, o bien dinámicamente, utilizando la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite "vivificar" los indicadores, para que cambien su apariencia en función de la situación actual.

Los segmentos se trazan desde un valor no vacío de un búfer hasta otro valor no vacío de otro búfer indicador. Para indicar qué valor se debe considerar "vacío", establezca este valor en la propiedad PLOT_EMPTY_VALUE:

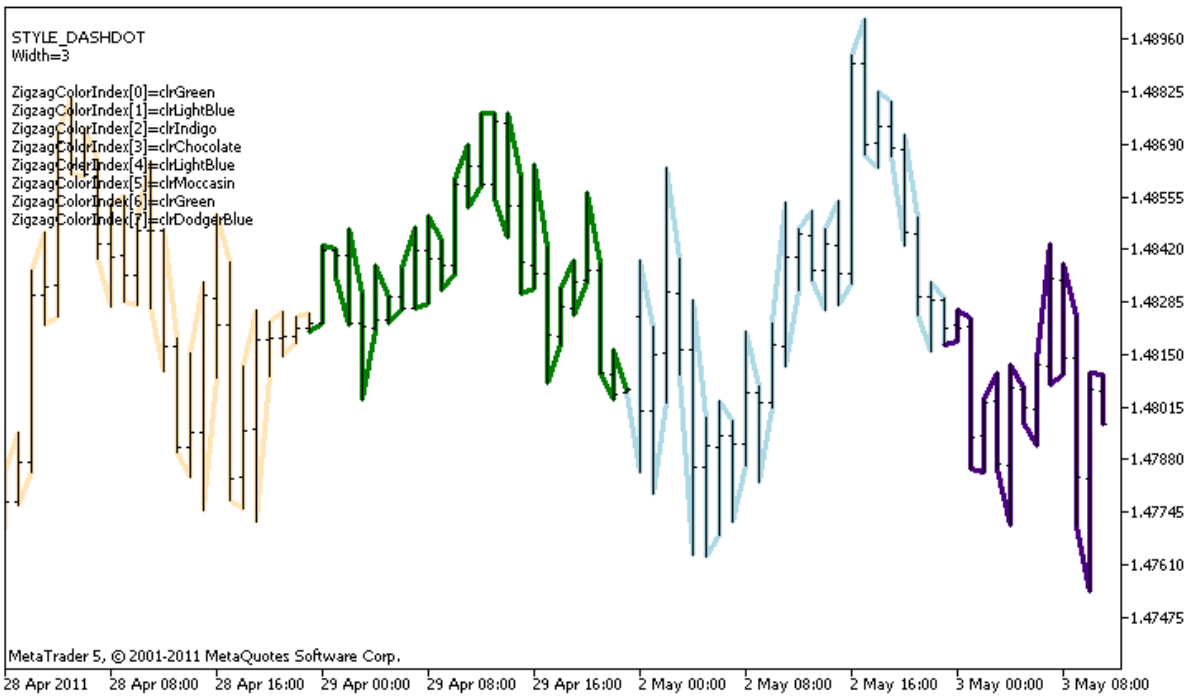
```
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(índice_ de_construcción_DRAW_COLOR_ZIGZAG,PLOT_EMPTY_VALUE,0);
```

Rellene siempre los búfers indicadores con valores de forma explícita, para las barras que se ignoran indique en el búfer un valor vacío.

El número de búfers requeridos para construir DRAW_COLOR_ZIGZAG — 3:

- dos búfers para almacenar los valores de los extremos de un segmento del zigzag;
- un búfer para almacenar el índice de color con el que se colorea el segmento (tiene sentido establecer sólo para los valores no vacíos).

Aquí tenemos un ejemplo del indicador que traza la sierra a base de los precios High y Low. El color, grosor y el estilo de la línea del zigzag se cambian de forma aleatoria cada N tics.



Fíjense, inicialmente para la construcción gráfica **plot1** con el estilo DRAW_COLOR_ZIGZAG se establecen 8 colores mediante la directiva del compilador #property, y luego en la función OnCalculate () el color se elige aleatoriamente de 14 colores que se guardan en el array colors[].

El parámetro N está pasado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

```
//+------------------------------------------------------------------+
//|                                              DRAW_COLOR_ZIGZAG.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_COLOR_ZIGZAG"
#property description "Dibuja una línea quebrada con segmentos de colores, el color d
#property description "El color, grosor y el estilo de segmentos se cambia de forma a
#property description " dentro de cada N tics"

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   1
//--- plot Color_Zigzag
#property indicator_label1  "Color_Zigzag"
#property indicator_type1   DRAW_COLOR_ZIGZAG
//--- estableceremos 8 colores para colorear los segmentos (se guardan en un array es
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- parámetro input
input int       N=5;              // número de tics a cambiar
int             color_sections;
//--- búfers de valores de los extremos de segmentos
double          Color_ZigzagBuffer1[];
double          Color_ZigzagBuffer2[];
//--- búfer de los índices de color para los extremos de los segmentos
double          Color_ZigzagColors[];
//--- el array para almacenar colores tiene 14 elementos
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrWhiteSmoke,clrCyan,clrMediumPur
  };
//--- array para almacenar estilos de trazado de la línea
ENUM_LINE_STYLE styles[]={STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDO
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
```

```
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,Color_ZigzagBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,Color_ZigzagBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,Color_ZigzagColors,INDICATOR_COLOR_INDEX);
//---- número de colores para colorear el zigzag
   color_sections=8;   //  ver comentario para la propiedad #property indicator_color
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la línea
   ticks++;
//--- si tenemos acumulado un número suficiente de tics
   if(ticks>=N)
     {
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- cambiamos colores con los que se dibujan segmentos
      ChangeColors(colors,color_sections);
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }

//--- necesitaremos una estructura de tiempo para obtener el día de la semana de cada
   MqlDateTime dt;

//--- posición de inicio del cálculo
   int start=0;
//--- si el indicador se calcula en el tic anterior, empezamos el cálculo a partir de
   if(prev_calculated!=0) start=prev_calculated-1;
//--- ciclo de cálculos
```

```
   for(int i=start;i<rates_total;i++)
     {
      //--- apuntaremos en la estructura la hora de apertura de la barra
      TimeToStruct(time[i],dt);

      //--- si el número de la barra es par
      if(i%2==0)
        {
         //---  escribimos en el 1-r búfer High, en el 2-do Low
         Color_ZigzagBuffer1[i]=high[i];
         Color_ZigzagBuffer2[i]=low[i];
         //--- color del segmento
         Color_ZigzagColors[i]=dt.day_of_year%color_sections;
        }
      //--- el número de la barra es impar
      else
        {
         //--- llenamos la barra en sentido inverso
         Color_ZigzagBuffer1[i]=low[i];
         Color_ZigzagBuffer2[i]=high[i];
         //--- color del segmento
         Color_ZigzagColors[i]=dt.day_of_year%color_sections;
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//|  cambia el color de segmentos del zigzag
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- número de colores
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- definimos de forma aleatoria un color nuevo para cada índice de colores
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- obtendremos un número aleatorio
      int number=MathRand();
      //--- obtendremos un índice en el array col[] como el remanente de la división
      int i=number%size;
      //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
      PlotIndexSetInteger(0,                    //  número del estilo gráfico
                          PLOT_LINE_COLOR,      //  identificador de la propiedad
                          plot_color_ind,       //  índice del color donde escribirem
```

```
                           cols[i]);              //  nuevo color
      //--- apuntaremos los colores
      comm=comm+StringFormat("ZigzagColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStr
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de segmentos en el zigzag                   |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de Color_ZigZag
   string comm="";
//--- bloque de cambio del grosor de la línea
   int number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5;   // el grosor puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- bloque de cambio del estilo de la línea
   number=MathRand();
//--- el divisor del número es igual al tamaño del array styles
   int size=ArraySize(styles);
//--- obtenemos el índice para seleccionar nuevo estilo como el remanente de la divis
   int style_index=number%size;
//--- estableceremos el color como la propiedad PLOT_LINE_COLOR
   PlotIndexSetInteger(0,PLOT_LINE_STYLE,styles[style_index]);
//--- apuntaremos el estilo de la línea
   comm="\r\n"+EnumToString(styles[style_index])+""+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_COLOR_BARS

El estilo DRAW_COLOR_BARS dibuja las barras basándose en los valores de cuatro búfers indicadores que contienen los precios Open, High, Low y Close. Este estilo es una versión más avanzada del estilo DRAW_BARS y permite establecer para cada barra su propio color desde un conjunto de colores predefinido previamente. Se utiliza para crear sus propios indicadores personalizados en forma de barras, también en otra subventana del gráfico y para otros instrumentos financieros.

El color de las barras se puede definir con las directivas del compilador, o dinámicamente a través de la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite "vivificar" los indicadores, para que cambien su apariencia en función de la situación actual.

El indicador se dibuja sólo para las barras que tienen establecidos los valores no vacíos de **todos** los cuatro búfers indicadores. Para indicar qué valor se debe considerar "vacío", establezca este valor en la propiedad PLOT_EMPTY_VALUE:

```
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(índice_ de_construcción_DRAW_COLOR_BARS,PLOT_EMPTY_VALUE,0);
```

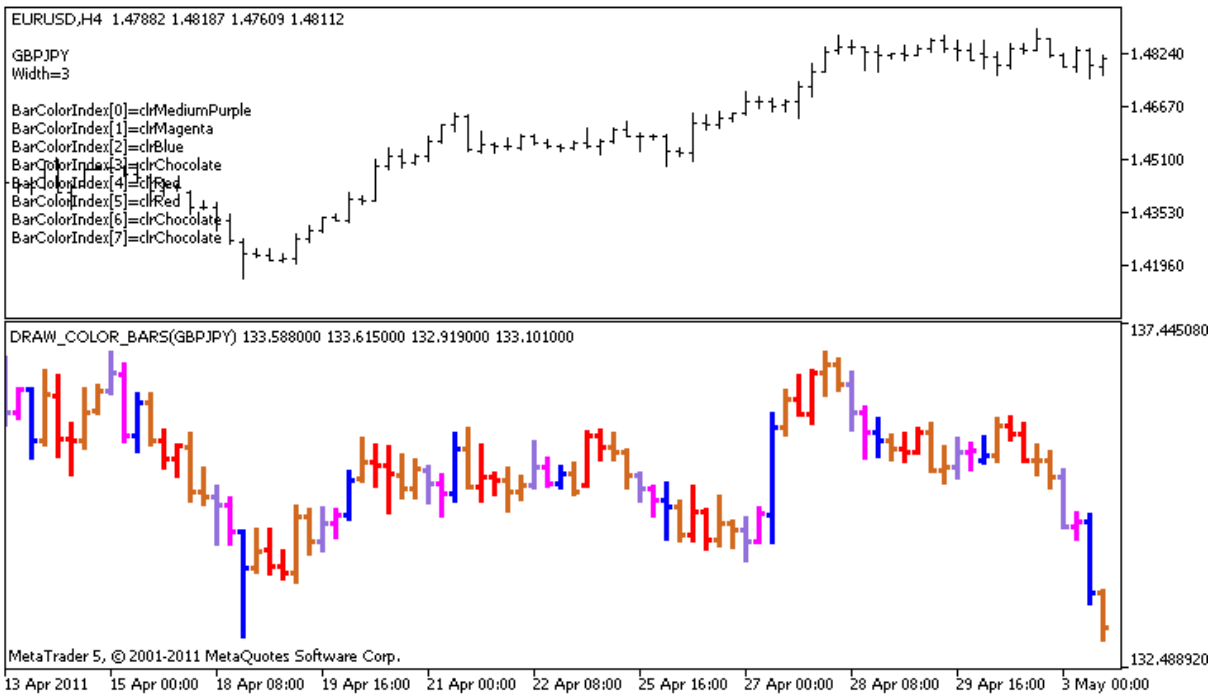
Rellene siempre los búfers indicadores con valores de forma explícita, para las barras que se ignoran indique en el búfer un valor vacío.

El número de búfers requeridos para construir DRAW_COLOR_BARS — 5:

- cuatro búfers para almacenar los valores Open, High, Low y Close;
- un búfer para almacenar el índice de color con el que se dibuja la barra (tiene sentido establecerlo sólo para las barras a dibujar).

Todos los búfers que se utilizan para la construcción deben ir en serie uno detrás del otro en orden establecido: Open, High, Low, Close y el búfer de color. Ninguno de los búfers de precios puede contener sólo los valores vacíos, porque en este caso no se dibuja nada.

Aquí tenemos un ejemplo del indicador que dibuja en una ventana separada las barras para el instrumento financiero especificado. El color de las barras se cambia aleatoriamente cada **N** tics. El parámetro N está sacado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).



Fíjense, inicialmente para la construcción gráfica **plot1** con el estilo DRAW_COLOR_BARS se establecen 8 colores mediante la directiva del compilador #property, y luego en la función OnCalculate () el color se elige aleatoriamente de 14 colores que se guardan en el array colors[].

```
//+------------------------------------------------------------------+
//|                                                DRAW_COLOR_BARS.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_COLOR_BARS"
#property description "Dibuja en la ventana separada las barras de diferentes colores
#property description "El color y el grosor de las barras, igual que el estilo, se ca
#property description "cada N tics"

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//--- plot ColorBars
#property indicator_label1  "ColorBars"
#property indicator_type1   DRAW_COLOR_BARS
//--- estableceremos 8 colores para colorear las barras (se guardan en un array espec
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- parámetros input
```

```
input int      N=5;              // número de tics para el cambio de apariencia
input int      bars=500;         // número de barras a mostrar
input bool     messages=false;   // mostrar mensajes en el log "Asesores Expertos"
//--- búfers indicadores
double         ColorBarsBuffer1[];
double         ColorBarsBuffer2[];
double         ColorBarsBuffer3[];
double         ColorBarsBuffer4[];
double         ColorBarsColors[];
//--- nombre del símbolo
string symbol;
int    bars_colors;
//--- el array para almacenar colores tiene 14 elementos
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
  };
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorBarsBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ColorBarsBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,ColorBarsBuffer3,INDICATOR_DATA);
   SetIndexBuffer(3,ColorBarsBuffer4,INDICATOR_DATA);
   SetIndexBuffer(4,ColorBarsColors,INDICATOR_COLOR_INDEX);
//---- número de colores para colorear las barras
   bars_colors=8;   //  ver comentario para la propiedad #property indicator_color1
//---
   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 ticks=0;
//--- contamos los tics para el cambio del estilo, color y grosor de la barra
   ticks++;
//--- si tenemos acumulado un número suficiente de tics
   if(ticks>=N)
     {
      //--- escogeremos nuevo símbolo en la ventana "Observación del mercado"
      symbol=GetRandomSymbolName();
      //--- cambiamos las propiedades de la línea
      ChangeLineAppearance();
      //--- cambiamos los colores con los que se dibujan las barras
      ChangeColors(colors,bars_colors);
      int tries=0;
      //--- haremos 5 intentos de llenar el búfer con los precios desde symbol
      while(!CopyFromSymbolToBuffers(symbol,rates_total,bars_colors) && tries<5)
        {
         //--- contador de llamadas a la función CopyFromSymbolToBuffers()
         tries++;
        }
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//|  Llenamos los búfers indicadores con los precios                 |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,int total,int bar_colors)
  {
//--- vamos a copiar los precios Open, High, Low y Close al array rates[]
   MqlRates rates[];
//--- contador de intentos
   int attempts=0;
//--- cantidad que se ha copiado ya
   int copied=0;
//--- hacemos 25 intentos de obtener la serie temporal para el símbolo necesario
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attem
     }
//--- si no se ha podido copiar la cantidad suficiente de barras
   if(copied!=bars)
     {
      //--- formaremos la cadena del mensaje
      string comm=StringFormat("Para el símbolo %s se ha logrado obtener sólo %d barr
```

```
                                name,
                                copied,
                                bars
                                );
      //--- mostraremos el mensaje en un comentario en la ventana principal del gráfi
      Comment(comm);
      //--- mostramos el mensaje
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- estableceremos la visualización del símbolo
      PlotIndexSetString(0,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low;"+name+"
      IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_BARS("+name+")");
     }
//--- inicializaremos los búfers con valores vacíos
   ArrayInitialize(ColorBarsBuffer1,0.0);
   ArrayInitialize(ColorBarsBuffer2,0.0);
   ArrayInitialize(ColorBarsBuffer3,0.0);
   ArrayInitialize(ColorBarsBuffer4,0.0);

//--- copiamos los precios a los búfers
   for(int i=0;i<copied;i++)
     {
      //--- calcularemos el índice correspondiente para los búfers
      int buffer_index=total-copied+i;
      //--- escribimos los precios en los búfers
      ColorBarsBuffer1[buffer_index]=rates[i].open;
      ColorBarsBuffer2[buffer_index]=rates[i].high;
      ColorBarsBuffer3[buffer_index]=rates[i].low;
      ColorBarsBuffer4[buffer_index]=rates[i].close;
      //---
      ColorBarsColors[buffer_index]=i%bar_colors;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| devuelve aleatoriamente el símbolo desde Market Watch            |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- número de símbolos mostrados en la ventana "Observación del mercado"
   int symbols=SymbolsTotal(true);
//--- posición del símbolo en la lista - un número aleatorio de 0 a symbols
   int number=MathRand()%symbols;
//--- devolvemos el nombre del símbolo sobre la posición indicada
   return SymbolName(number,true);
```

```
  }
//+------------------------------------------------------------------+
//|  cambia el color de segmentos del zigzag
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- número de colores
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- definimos de forma aleatoria un color nuevo para cada índice de colores
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- obtendremos un número aleatorio
      int number=MathRand();
      //--- obtendremos un índice en el array col[] como el remanente de la división
      int i=number%size;
      //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
      PlotIndexSetInteger(0,                    //  número del estilo gráfico
                          PLOT_LINE_COLOR,      //  identificador de la propiedad
                          plot_color_ind,       //  índice del color donde escribirem
                          cols[i]);             //  nuevo color
      //--- apuntaremos los colores
      comm=comm+StringFormat("BarColorIndex[%d]=%s \r\n",plot_color_ind,ColorToString
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de las barras                               |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de las barras
   string comm="";

//--- bloque del cambio del grosor de las barras
   int number=MathRand();
//--- obtenemos el grosor como el remanente de la división de números enteros
   int width=number%5;   // el grosor puede ser de 0 a 4
//--- estableceremos el color como la propiedad PLOT_LINE_WIDTH
   PlotIndexSetInteger(0,PLOT_LINE_WIDTH,width);
//--- apuntaremos el grosor de la línea
   comm=comm+"\r\nWidth="+IntegerToString(width);

//--- apuntamos el nombre del símbolo
   comm="\r\n"+symbol+comm;
```

```
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

DRAW_COLOR_CANDLES

El estilo DRAW_COLOR_CANDLES, igual que el DRAW_CANDLES, dibuja las velas japonesas basándose en los valores de cuatro búfers indicadores que contienen los precios Open, High, Low y Close. Pero además de eso, este estilo permite establecer un color para cada vela desde un conjunto predefinido. Para eso, en el estilo ha sido agregado un búfer especial de colores que guarda los índices de los colores para cada barra. Se utiliza para crear sus propios indicadores personalizados en forma de velas japonesas, también en otra subventana del gráfico y para otros instrumentos financieros.

Usted puede establecer el número de colores para colorear las velas a través de las directivas del compilador, o dinámicamente a través de la función PlotIndexSetInteger(). El cambio dinámico de las propiedades de la construcción gráfica permite "vivificar" los indicadores, para que cambien su apariencia en función de la situación actual.

El indicador se dibuja sólo para las barras que tienen establecidos los valores no vacíos en los cuatro búfers de precios. Para indicar qué valor se debe considerar "vacío", establezca este valor en la propiedad PLOT_EMPTY_VALUE:

```
//--- el valor 0 (valor vacío) no va a participar en el proceso de trazado
   PlotIndexSetDouble(índice_ de_construcción_DRAW_COLOR_CANDLES,PLOT_EMPTY_VALUE,0);
```

Rellene siempre los búfers indicadores con valores de forma explícita, para las barras que se ignoran indique en el búfer un valor vacío.

El número de búfers requeridos para construir DRAW_COLOR_CANDLES — 5:

- cuatro búfers para almacenar los valores Open, High, Low y Close;
- un búfer para almacenar el índice de color con el que se dibuja la vela (tiene sentido establecerlo sólo para las velas a dibujar).

Todos los búfers que se utilizan para la construcción deben ir en serie uno detrás del otro en orden establecido: Open, High, Low, Close y el búfer de color. Ninguno de los búfers de precios puede contener sólo los valores vacíos, porque en este caso no se dibuja nada.

Aquí tenemos un ejemplo del indicador que dibuja en una ventana separada las velas japonesas para el instrumento financiero especificado. El color de las velas se cambia aleatoriamente cada **N** tics. El parámetro N está sacado a los parámetros externos del indicador para que exista la posibilidad de establecerlo manualmente (pestaña "Parámetros" en la ventana de propiedades del indicador).

Fíjense en que inicialmente para la construcción gráfica **plot1** el color se establece mediante la directiva del compilador #property, y luego en la funciónOnCalculate() se elige aleatoriamente un nuevo color desde la lista previamente preparada.

```
//+------------------------------------------------------------------+
//|                                            DRAW_COLOR_CANDLES.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

#property description "Indicador para demostrar DRAW_COLOR_CANDLES."
#property description "Dibuja en la ventana separada las velas para el símbolo selecc
#property description " "
#property description "El color y el grosor de las velas, igual que el estilo, se cam
#property description "de forma aleatoria cada N tics."

#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//--- plot ColorCandles
#property indicator_label1  "ColorCandles"
#property indicator_type1   DRAW_COLOR_CANDLES
//--- estableceremos 8 colores para colorear las velas (se guardan en un array especi
#property indicator_color1  clrRed,clrBlue,clrGreen,clrYellow,clrMagenta,clrCyan,clrL
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

```
//--- parámetros input
input int      N=5;              // número de tics para el cambio de apariencia
input int      bars=500;         // número de velas a mostrar
input bool     messages=false;   // mostrar mensajes en el log "Asesores Expertos"
//--- búfers indicadores
double         ColorCandlesBuffer1[];
double         ColorCandlesBuffer2[];
double         ColorCandlesBuffer3[];
double         ColorCandlesBuffer4[];
double         ColorCandlesColors[];
int            candles_colors;
//--- nombre del símbolo
string symbol;
//--- el array para almacenar colores tiene 14 elementos
color colors[]=
  {
   clrRed,clrBlue,clrGreen,clrChocolate,clrMagenta,clrDodgerBlue,clrGoldenrod,
   clrIndigo,clrLightBlue,clrAliceBlue,clrMoccasin,clrMagenta,clrCyan,clrMediumPurple
  };
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- si bars es muy pequeño - finalizamos el trabajo antes de tiempo
   if(bars<50)
     {
      Comment("¡Por favor, indique el número más grande de barras! El trabajo del ind
      return(INIT_PARAMETERS_INCORRECT);
     }
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorCandlesBuffer1,INDICATOR_DATA);
   SetIndexBuffer(1,ColorCandlesBuffer2,INDICATOR_DATA);
   SetIndexBuffer(2,ColorCandlesBuffer3,INDICATOR_DATA);
   SetIndexBuffer(3,ColorCandlesBuffer4,INDICATOR_DATA);
   SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
//--- valor vacío
   PlotIndexSetDouble(0,PLOT_EMPTY_VALUE,0);
//--- nombre del símbolo para el que se dibujan las barras
   symbol=_Symbol;
//--- estableceremos la visualización del símbolo
   PlotIndexSetString(0,PLOT_LABEL,symbol+" Open;"+symbol+" High;"+symbol+" Low;"+sym
   IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES("+symbol+")");
//---- número de colores para colorear las velas
   candles_colors=8;      //  ver comentario para la propiedad #property indicator_col
//---
   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 ticks=INT_MAX-100;
//--- contamos los tics para el cambio del estilo y color
   ticks++;
//--- si tenemos acumulado un número suficiente de tics
   if(ticks>=N)
     {
      //--- escogeremos nuevo símbolo en la ventana "Observación del mercado"
      symbol=GetRandomSymbolName();
      //--- cambiamos la apariencia
      ChangeLineAppearance();
      //--- cambiamos los colores con los que se dibujan las barras
      ChangeColors(colors,candles_colors);

      int tries=0;
      //--- haremos 5 intentos de llenar el búfer plot1 con los precios desde symbol
      while(!CopyFromSymbolToBuffers(symbol,rates_total,0,
            ColorCandlesBuffer1,ColorCandlesBuffer2,ColorCandlesBuffer3,
            ColorCandlesBuffer4,ColorCandlesColors,candles_colors)
            && tries<5)
        {
         //--- contador de llamadas a la función CopyFromSymbolToBuffers()
         tries++;
        }
      //--- actualizamos el contador de tics pasándolo a cero
      ticks=0;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//|  Llena la vela indicada                                          |
//+------------------------------------------------------------------+
bool CopyFromSymbolToBuffers(string name,
```

```
                              int total,
                              int plot_index,
                              double &buff1[],
                              double &buff2[],
                              double &buff3[],
                              double &buff4[],
                              double &col_buffer[],
                              int    cndl_colors
                              )
  {
//--- vamos a copiar los precios Open, High, Low y Close al array rates[]
   MqlRates rates[];
//--- contador de intentos
   int attempts=0;
//--- cantidad que se ha copiado ya
   int copied=0;
//--- hacemos 25 intentos de obtener la serie temporal para el símbolo necesario
   while(attempts<25 && (copied=CopyRates(name,_Period,0,bars,rates))<0)
     {
      Sleep(100);
      attempts++;
      if(messages) PrintFormat("%s CopyRates(%s) attempts=%d",__FUNCTION__,name,attem
     }
//--- si no se ha podido copiar la cantidad suficiente de barras
   if(copied!=bars)
     {
      //--- formaremos la cadena del mensaje
      string comm=StringFormat("Para el símbolo %s se ha logrado obtener sólo %d barr
                               name,
                               copied,
                               bars
                               );
      //--- mostraremos el mensaje en un comentario en la ventana principal del gráfi
      Comment(comm);
      //--- mostramos el mensaje
      if(messages) Print(comm);
      return(false);
     }
   else
     {
      //--- estableceremos la visualización del símbolo
      PlotIndexSetString(plot_index,PLOT_LABEL,name+" Open;"+name+" High;"+name+" Low
      IndicatorSetString(INDICATOR_SHORTNAME,"DRAW_COLOR_CANDLES("+symbol+")");
     }
//--- inicializaremos los búfers con valores vacíos
   ArrayInitialize(buff1,0.0);
   ArrayInitialize(buff2,0.0);
   ArrayInitialize(buff3,0.0);
```

```
   ArrayInitialize(buff4,0.0);
//--- en cada tic copiamos los precios a los búfers
   for(int i=0;i<copied;i++)
     {
      //--- calcularemos el índice correspondiente para los búfers
      int buffer_index=total-copied+i;
      //--- escribimos los precios en los búfers
      buff1[buffer_index]=rates[i].open;
      buff2[buffer_index]=rates[i].high;
      buff3[buffer_index]=rates[i].low;
      buff4[buffer_index]=rates[i].close;
      //--- estableceremos el color de la vela
      int color_index=i%cndl_colors;
      col_buffer[buffer_index]=color_index;
     }
   return(true);
  }
//+------------------------------------------------------------------+
//| devuelve aleatoriamente el símbolo desde Market Watch            |
//+------------------------------------------------------------------+
string GetRandomSymbolName()
  {
//--- número de símbolos mostrados en la ventana "Observación del mercado"
   int symbols=SymbolsTotal(true);
//--- posición del símbolo en la lista - un número aleatorio de 0 a symbols
   int number=MathRand()%symbols;
//--- devolvemos el nombre del símbolo sobre la posición indicada
   return SymbolName(number,true);
  }
//+------------------------------------------------------------------+
//|  cambia el color de segmentos de las velas
//+------------------------------------------------------------------+
void  ChangeColors(color  &cols[],int plot_colors)
  {
//--- número de colores
   int size=ArraySize(cols);
//---
   string comm=ChartGetString(0,CHART_COMMENT)+"\r\n\r\n";

//--- definimos de forma aleatoria un color nuevo para cada índice de colores
   for(int plot_color_ind=0;plot_color_ind<plot_colors;plot_color_ind++)
     {
      //--- obtendremos un número aleatorio
      int number=MathRand();
      //--- obtendremos un índice en el array col[] como el remanente de la división
      int i=number%size;
      //--- estableceremos el color para cada índice como la propiedad PLOT_LINE_COLO
      PlotIndexSetInteger(0,                    //  número del estilo gráfico
```

```
                           PLOT_LINE_COLOR,      //  identificador de la propiedad
                           plot_color_ind,       //  índice del color donde escribirem
                           cols[i]);             //  nuevo color
      //--- apuntaremos los colores
      comm=comm+StringFormat("CandleColorIndex[%d]=%s \r\n",plot_color_ind,ColorToStr
      ChartSetString(0,CHART_COMMENT,comm);
     }
//---
  }
//+------------------------------------------------------------------+
//| cambia la apariencia de las velas                                |
//+------------------------------------------------------------------+
void ChangeLineAppearance()
  {
//--- cadena para formar la información sobre las propiedades de las velas
   string comm="";
//--- apuntamos el nombre del símbolo
   comm="\r\n"+symbol+comm;
//--- mostraremos la información en el gráfico a través del comentario
   Comment(comm);
  }
```

Relación entre las propiedades de indicador y funciones correspondientes

Cada uno de los indicadores personalizados tiene una gran cantidad de propiedades una parte de las cuales es obligatoria y siempre se encuentra al principio de la descripción. Estas propiedades son las siguientes:

- indicación de la ventana para mostrar el indicador – indicator_separate_window o indicator_chart_window;
- cantidad de los búfers de indicador – indicator_buffers;
- cantidad de construcciones (representaciones) gráficas que muestra el indicador – indicator_plots.

Pero hay otras propiedades que pueden ser definidas a través de las directivas del preprocesador y mediante las funciones para la creación del indicador personalizado. En la tabla de abajo se enumeran estas propiedades y sus funciones correspondientes.

Directivas para las propiedades de subventana del indicador	Funciones del tipo IndicatorSet...()	Descripción de la propiedad ajustada de subventana
indicator_height	IndicatorSetInteger (INDICATOR_INDICATOR_HEIGHT, nHeight)	Valor del alto fijo de la subventana
indicator_minimum	IndicatorSetDouble (INDICATOR_MINIMUM, dMaxValue)	Valor mínimo del eje vertical
indicator_maximum	IndicatorSetDouble (INDICATOR_MAXIMUM, dMinValue)	Valor máximo del eje vertical
indicator_levelN	IndicatorSetDouble (INDICATOR_LEVELVALUE, N-1, nLevelValue)	Valor del eje vertical para el nivel N
no hay directiva del preprocesador	IndicatorSetString (INDICATOR_LEVELTEXT, N-1, sLevelName)	Nombre del nivel mostrado
indicator_levelcolor	IndicatorSetInteger (INDICATOR_LEVELCOLOR, N-1, nLevelColor)	Color del nivel N
indicator_levelwidth	IndicatorSetInteger (INDICATOR_LEVELWIDTH, N-1, nLevelWidth)	Grosor de línea del nivel N
indicator_levelstyle	IndicatorSetInteger (INDICATOR_LEVELSTYLE, N-1, nLevelStyle)	Estilo de línea del nivel N
Directrices para las propiedades de	**Funciones del tipo PlotIndexSet...()**	**Descripción de la propiedad establecida para una**

representaciones gráficas		representación gráfica
indicator_labelN	PlotIndexSetString(N-1,PLOT_LABEL,sLabel)	Denominación breve para representación gráfica N. Se muestra en la ventana DataWindow y en la ayuda contextual al apuntar con el cursor en el gráfico
indicator_colorN	PlotIndexSetInteger(N-1, PLOT_LINE_COLOR, nColor)	Color de línea para representación gráfica N
indicator_styleN	PlotIndexSetInteger(N-1, PLOT_LINE_STYLE, nType)	Estilo de línea para representación gráfica N
indicator_typeN	PlotIndexSetInteger(N-1, PLOT_DRAW_TYPE, nType)	Tipo de línea para representación gráfica N
indicator_widthN	PlotIndexSetInteger(N-1, PLOT_LINE_WIDTH, nWidth)	Grosor de línea para representación gráfica N
Propiedades generales de indicador	**Funciones del tipo IndicatorSet...()**	**Descripción**
no hay directiva del preprocesador	IndicatorSetString (INDICATOR_SHORTNAME, sShortName)	Establece un nombre breve y cómodo del indicador que va a visualizarse en la lista de indicadores (se llama en el terminal con la combinación **Ctrl+I**).
no hay directiva del preprocesador	IndicatorSetInteger (INDICATOR_DIGITS, nDigits)	Establece la precisión necesaria para visualizar valores del indicador, es decir, el número de dígitos después del punto decimal
no hay directiva del preprocesador	IndicatorSetInteger (INDICATOR_LEVELS, nLevels)	Establece la cantidad de niveles en la ventana del indicador
indicator_applied_price	No hay función, la propiedad se establece sólo con la directriz del preprocesador.	Tipo de precio por defecto que se utiliza para calcular el valor del indicador. Se especifica si hace falta sólo si se utiliza la función del primer tipo OnCalculate(). Valor de la propiedad puede ser establecido desde el diálogo de propiedades del indicador en la pestaña "Parámetros" - "Aplicar a".

Cabe destacar que la numeración de niveles y representaciones gráficas en términos de preprocesador se empieza desde uno, mientras que la numeración de las mismas propiedades cuando se usan las

funciones se empieza desde cero, es decir, hay que indicar el valor 1 menos que indicaríamos utilizando #property.

Hay algunas directrices para las que no existen funciones correspondientes:

Directriz	Descripción
indicator_chart_window	Indicador se visualiza en la ventana principal
indicator_separate_window	Indicador se visualiza en la subventana separada
indicator_buffers	Indica la cantidad necesaria de buffers requeridos
indicator_plots	Indica la cantidad de representaciones gráficas en el indicador

SetIndexBuffer

Enlaza el búfer de indicador especificado con el array dinámico unidimensional del tipo double.

```
bool  SetIndexBuffer(
   int                   index,         // índice del buffer
   double                buffer[],      // array
   ENUM_INDEXBUFFER_TYPE  data_type      // lo que vamos a almacenar
   );
```

Parámetros

index

[in] Número del búfer de indicadores. La numeración se empieza desde 0. El número tiene que ser menos que el valor declarado en #property indicator_buffers.

buffer[]

[in] Array declarado en el programa del indicador personalizado.

data_type

[in] Tipo de datos guardados en el array de indicador. Por defecto INDICATOR_DATA (valores del indicador calculado). También puede adquirir el valor INDICATOR_COLOR_INDEX, entonces este búfer sirve para almacenar los índices de colores para el anterior búfer de indicadores. Se puede establecer hasta 64 colores en la línea #property indicator_colorN. El valor INDICATOR_CALCULATIONS significa que este buffer se usa en los cálculos intermedios del indicador y no está destinado para dibujar.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false.

Nota

Después de enlazado el array dinámico *buffer[]* tendrá la indexación como los arrays comunes, incluso si para este array previamente ha sido establecida la indexación como en las series temporales. Si hace falta cambiar el orden de acceso a los elementos del array de indicador, es necesario usar la función ArraySetAsSeries() después de enlazar el array usando la función SetIndexBuffer(). Además, hay que tener en cuenta que no se puede cambiar el tamaño de los arrays dinámicos que han sido asignados como búfers de indicadores por la función SetIndexBuffer (). Todas las operaciones relacionadas con el cambio de tamaño de los búfers de indicadores se realizan por el subsistema ejecutivo del terminal.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                TestCopyBuffer1.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
```

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---- plot MA
#property indicator_label1  "MA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- input parameters
input bool               AsSeries=true;
input int                period=15;
input ENUM_MA_METHOD     smootMode=MODE_EMA;
input ENUM_APPLIED_PRICE price=PRICE_CLOSE;
input int                shift=0;
//--- indicator buffers
double                   MABuffer[];
int                      ma_handle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- indicator buffers mapping
   if(AsSeries) ArraySetAsSeries(MABuffer,true);
   Print("Búfer de indicadores es una serie temporal = ",ArrayGetAsSeries(MABuffer));
   SetIndexBuffer(0,MABuffer,INDICATOR_DATA);
   Print("Búfer de indicadores después de SetIndexBuffer() es una serie temporal = ",
         ArrayGetAsSeries(MABuffer));

//--- vamos a cambiar el orden de acceso a los elementos del búfer de indicadores
   ArraySetAsSeries(MABuffer,AsSeries);

   IndicatorSetString(INDICATOR_SHORTNAME,"MA("+period+")"+AsSeries);
//---
   ma_handle=iMA(Symbol(),0,period,shift,smootMode,price);
   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[])
  {
//--- vamos a copiar los valores del promedio móvil en el buffer MABuffer
   int copied=CopyBuffer(ma_handle,0,0,rates_total,MABuffer);

   Print("MABuffer[0] = ",MABuffer[0]);// dependiendo del valor AsSeries
                                       // vamos a recibir el valor más antiguo
                                       // o en la barra actual no finalizada

//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

Véase también

Propiedades de indicadores personalizados, Acceso a las series temporales e indicadores

IndicatorSetDouble

Establece el valor de la propiedad correspondiente del indicador. La propiedad del indicador tiene que ser del tipo double. Existen 2 variantes de la función.

Llamada con especificación del identificador de la propiedad.

```
bool  IndicatorSetDouble(
   int     prop_id,           // identificador
   double  prop_value         // valor que se establece
   );
```

Llamada con especificación del identificador y modificador de la propiedad.

```
bool  IndicatorSetDouble(
   int     prop_id,           // identificador
   int     prop_modifier,     // modificador
   double  prop_value         // valor que se establece
   )
```

Parámetros

prop_id

[in] Identificador de la propiedad del indicador. Su valor puede ser uno de los valores de la enumeración ENUM_CUSTOMIND_PROPERTY_DOUBLE.

prop_modifier

[in] Modificador de la propiedad especificada. Sólo las propiedades de niveles requieren un modificador. La numeración de los niveles se empieza desde 0. Eso quiere decir que hay que especificar un uno para establecer la propiedad para el segundo nivel (uno menos que cuando se utiliza una directiva del compilador).

prop_value

[in] Valor de la propiedad.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false.

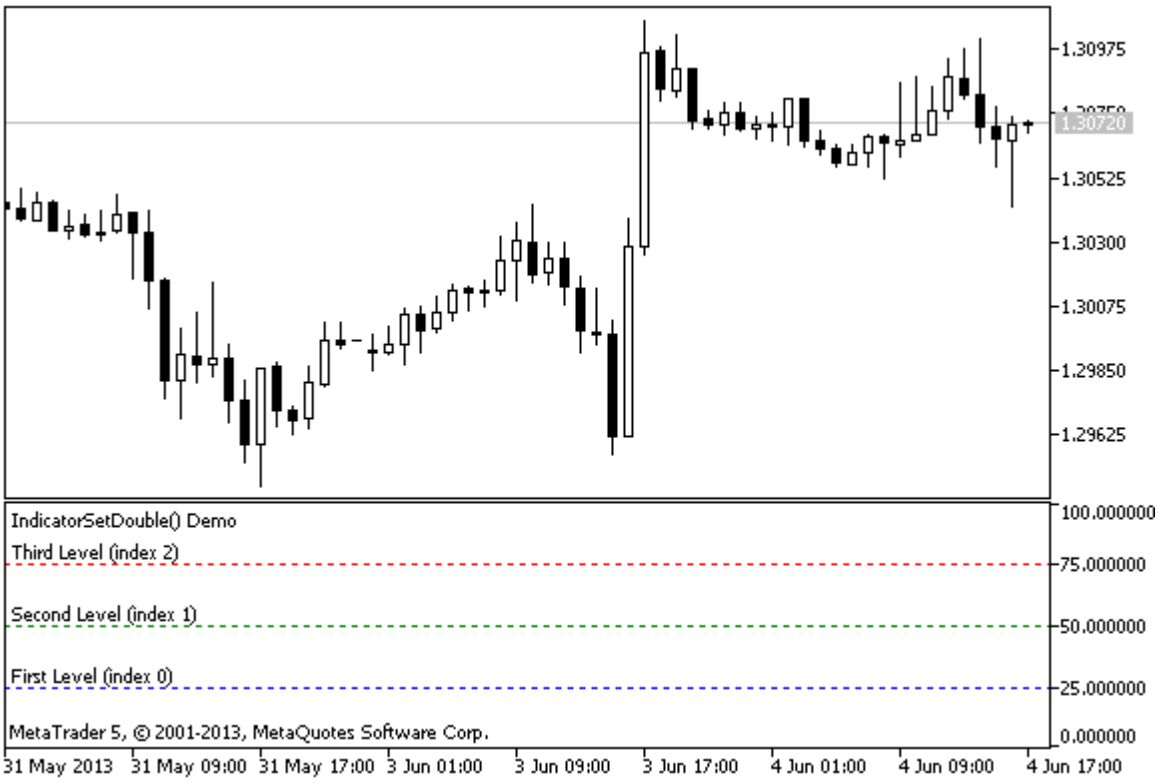
Nota

Cuando se utiliza la directiva #property la numeración de las propiedades (modificadores) se empieza desde 1 (un uno). Mientras que la función utiliza la numeración desde 0 (cero). Si el número del nivel no se establece de forma correcta, la visualización del indicador puede ser diferente de lo que está previsto.

Por ejemplo, hay dos maneras de establecer el valor del primer nivel para el indicador en una subventana separada:

- property indicator_level**1** 50 - se utiliza un 1 par especificar el número del nivel,
- IndicatorSetDouble(INDICATOR_LEVELVALUE, **0**, 50) - se utiliza 0 para especificar el primer nivel.

Пример: индикатор-"перевертыш", меняющий максимальное и минимальное значения окна индикатора, а также значения уровней, на которых расположены горизонтальные линии.



```
//--- установим максимальное и минимальное значения для окна индикатора
#property indicator_minimum  0
#property indicator_maximum  100
//--- зададим показ трех горизонтальных уровней в отдельном окне индикатора
#property indicator_level1 25
#property indicator_level2 50
#property indicator_level3 75
//--- установим толщину горизонтальных уровней
#property indicator_levelwidth 1
//--- установим стиль горизонтальных уровней
#property indicator_levelstyle STYLE_DOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- зададим описания горизонтальных уровней
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- зададим короткое имя индикатора
   IndicatorSetString(INDICATOR_SHORTNAME,"IndicatorSetDouble() Demo");
//--- зададим каждому уровню свой цвет
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,0,clrBlue);
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,1,clrGreen);
   IndicatorSetInteger(INDICATOR_LEVELCOLOR,2,clrRed);
//---
   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=50,level3=75;
   static double max=100,min=0, shift=100;
//--- считаем тики
   tick_counter++;
//--- на каждом 10-м тике делаем переворот
   if(tick_counter%10==0)
     {
      //--- перевернем знак для значений уровня
      level1=-level1;
      level2=-level2;
      level3=-level3;
      //--- перевернем знак для значений максимума и минимума
      max-=shift;
      min-=shift;
      //--- перевернем значение сдивга
      shift=-shift;
      //--- зададим новые значения уровней
```

```
      IndicatorSetDouble(INDICATOR_LEVELVALUE,0,level1);
      IndicatorSetDouble(INDICATOR_LEVELVALUE,1,level2);
      IndicatorSetDouble(INDICATOR_LEVELVALUE,2,level3);
      //--- зададим новые значения максимума и минимума для окна индикатора
      Print("Set up max = ",max,",   min = ",min);
      IndicatorSetDouble(INDICATOR_MAXIMUM,max);
      IndicatorSetDouble(INDICATOR_MINIMUM,min);
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
```

Véase también

Estilos de indicadores en ejemplos, Relación entre propiedades de indicador y funciones, Estilos de dibujo

IndicatorSetInteger

Establece el valor de la propiedad correspondiente del indicador. La propiedad del indicador tiene que ser del tipo int o color. Existen 2 variantes de la función.

Llamada con especificación del identificador de la propiedad.

```
bool  IndicatorSetInteger(
   int  prop_id,           // identificador
   int  prop_value         // valor que se establece
   );
```

Llamada con especificación del identificador y modificador de la propiedad.

```
bool  IndicatorSetInteger(
   int  prop_id,           // identificador
   int  prop_modifier,     // modificador
   int  prop_value         // valor que se establece
   )
```

Parámetros

prop_id

[in] Identificador de la propiedad del indicador. Su valor puede ser uno de los valores de la enumeración ENUM_CUSTOMIND_PROPERTY_INTEGER.

prop_modifier

[in] Modificador de la propiedad especificada. Sólo las propiedades de niveles requieren un modificador.

prop_value

[in] Valor de la propiedad.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false.

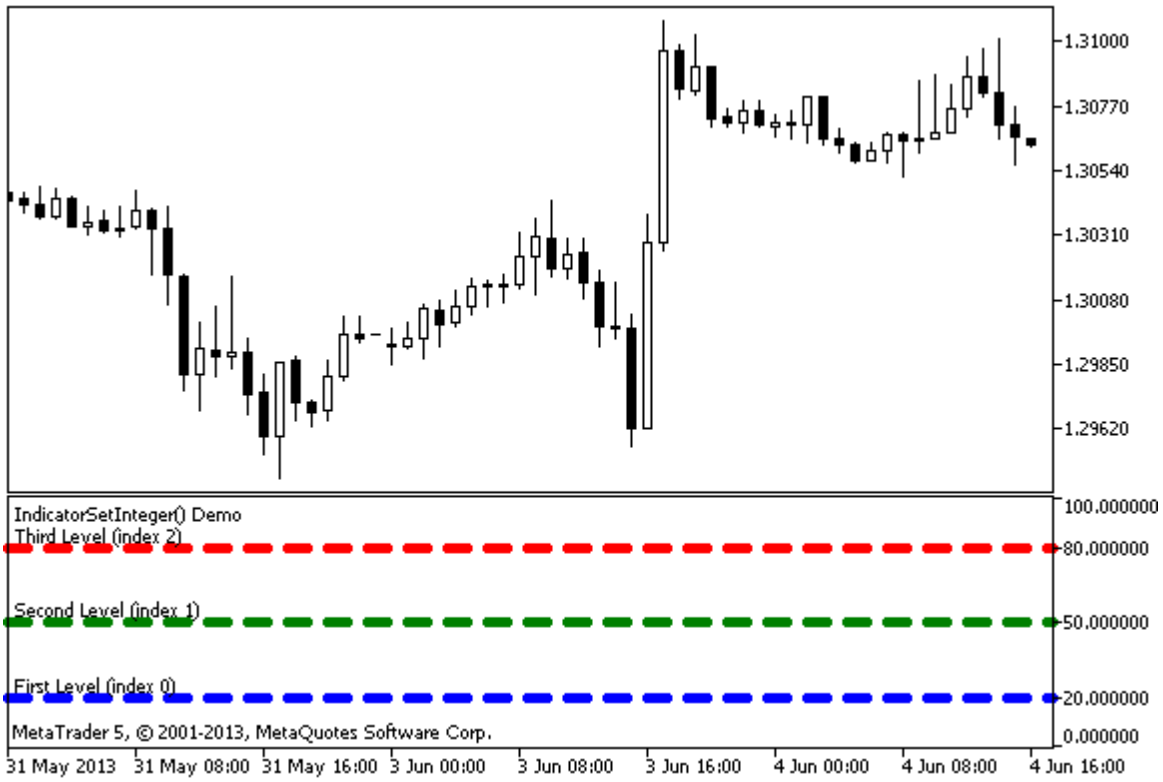
Примечание

Нумерация свойств (модификаторов) при использовании директивы #property начинается с 1 (единицы), в то время как функция использует нумерацию с 0 (нуля). При неправильном задании номера уровня отображение индикатора может отличаться от того, которое предполагается.

Например, чтобы задать толщину линии первого горизонтального уровня используйте нулевой индекс:

- IndicatorSetInteger(INDICATOR_LEVELWIDTH, **0**, 5) - используется индекс 0 для задания толщины линии первого уровня.

Пример: индикатор, задающий цвет, стиль и толщину горизонтальных уровней индикатора.



```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- зададим показ трех горизонтальных уровней в отдельном окне индикатора
#property indicator_level1 20
#property indicator_level2 50
#property indicator_level3 80
//--- установим толщину горизонтальных уровней
#property indicator_levelwidth 5
//--- установим цвет горизонтальных уровней
#property indicator_levelcolor clrAliceBlue
//--- установим стиль горизонтальных уровней
#property indicator_levelstyle STYLE_DOT
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- зададим описания горизонтальных уровней
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- зададим короткое имя индикатора
   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;
//--- считаем тики
   tick_counter++;
//--- и устанавливаем цвета горизонтальных уровней в зависимости от счетчика тиков
   ChangeLevelColor(0,tick_counter,3,6,10); // три последних параметра переключают цв
   ChangeLevelColor(1,tick_counter,3,6,8);
   ChangeLevelColor(2,tick_counter,4,7,9);
//--- поменяем стили горизонтальных уровней
   ChangeLevelStyle(0,tick_counter);
   ChangeLevelStyle(1,tick_counter+5);
   ChangeLevelStyle(2,tick_counter+15);
//--- получим толщину линии как остаток цедочисленного деления числа тиков на 5
   int width=tick_counter%5;
//--- пройдем по всем горизонтальным уровням и выставим
   for(int l=0;l<3;l++)
      IndicatorSetInteger(INDICATOR_LEVELWIDTH,l,width+1);
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Установим цвет горизонтальной линии в отдельном окне индикатора  |
```

```
//+------------------------------------------------------------------+
void ChangeLevelColor(int level,      // номер горизонтальной линии
                      int tick_number,// делимое, число для получения остатка деления
                      int f_trigger,  // первый делитель переключения цвета
                      int s_trigger,  // второй делитель переключения цвета
                      int t_trigger)  // третий делитель переключения цвета
  {
   static color colors[3]={clrRed,clrBlue,clrGreen};
//--- индекс цвета из массива colors[]
   int index=-1;
//--- вычислим номер цвета из массива colors[] для раскрашивания горизонтальной линии
   if(tick_number%f_trigger==0)
      index=0;   // если число tick_number делится без остатка на f_trigger
   if(tick_number%s_trigger==0)
      index=1;   // если число tick_number делится без остатка на s_trigger
   if(tick_number%t_trigger==0)
      index=2;   // если число tick_number делится без остатка на t_trigger
//--- если цвет определен, установим его
   if(index!=-1)
      IndicatorSetInteger(INDICATOR_LEVELCOLOR,level,colors[index]);
//---
  }
//+------------------------------------------------------------------+
//| Установим стиль горизонтальной линии в отдельном окне индикатора |
//+------------------------------------------------------------------+
void ChangeLevelStyle(int level,     // номер горизонтальной линии
                      int tick_number// число для получения остатка от деления)
                      )
  {
//--- массив для хранения стилей
   static ENUM_LINE_STYLE styles[5]=
     {STYLE_SOLID,STYLE_DASH,STYLE_DOT,STYLE_DASHDOT,STYLE_DASHDOTDOT};
//--- индекс стиля из массива styles[]
   int index=-1;
//--- вычислим номер из массива styles[] для установки стиля горизонтальной линии
   if(tick_number%50==0)
      index=5;   // если tick_number делится без остатка на 50, то стиль STYLE_DASHDO
   if(tick_number%40==0)
      index=4;   // ... стиль 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
//--- если стиль определен, установим его
   if(index!=-1)
      IndicatorSetInteger(INDICATOR_LEVELSTYLE,level,styles[index]);
  }
```

Смотри также

Свойства пользовательских индикаторов, Свойства программ (#property), Estilos de dibujo

IndicatorSetString

Establece el valor de la propiedad correspondiente del indicador. La propiedad del indicador tiene que ser del tipo string. Existen 2 variantes de la función.

Llamada con especificación del identificador de la propiedad.

```
bool  IndicatorSetString(
   int     prop_id,           // identificador
   string  prop_value         // valor que se establece
   );
```

Llamada con especificación del identificador y modificador de la propiedad.

```
bool  IndicatorSetString(
   int     prop_id,           // identificador
   int     prop_modifier,     // modificador
   string  prop_value         // valor que se establece
   )
```

Parámetros

prop_id

[in] Identificador de la propiedad del indicador. Su valor puede ser uno de los valores de la enumeración ENUM_CUSTOMIND_PROPERTY_STRING.

prop_modifier

[in] Modificador de la propiedad especificada. Sólo las propiedades de niveles requieren un modificador.

prop_value

[in] Valor de la propiedad.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false.

Примечание

Нумерация свойств (модификаторов) при использовании директивы #property начинается с 1 (единицы), в то время как функция использует нумерацию с 0 (нуля). При неправильном задании номера уровня отображение индикатора может отличаться от того, которое предполагается.

Например, чтобы задать описание первого горизонтального уровня используйте нулевой индекс:

- IndicatorSetString(INDICATOR_LEVELTEXT, **0**, "First Level") - используется индекс 0 для задания текстового описания первого уровня.

Пример: индикатор, устанавливающий подписи к горизонтальным уровням индикатора.


```
#property indicator_separate_window
#property indicator_minimum 0
#property indicator_maximum 100
//--- зададим показ трех горизонтальных уровней в отдельном окне индикатора
#property indicator_level1 30
#property indicator_level2 50
#property indicator_level3 70
//--- установим цвет горизонтальных уровней
#property indicator_levelcolor clrRed
//--- установим стиль горизонтальных уровней
#property indicator_levelstyle STYLE_SOLID
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- зададим описания горизонтальных уровней
   IndicatorSetString(INDICATOR_LEVELTEXT,0,"First Level (index 0)");
   IndicatorSetString(INDICATOR_LEVELTEXT,1,"Second Level (index 1)");
   IndicatorSetString(INDICATOR_LEVELTEXT,2,"Third Level (index 2)");
//--- зададим короткое имя индикатора
   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);
  }
```

Смотри также

Propiedades de indicadores personalizados, Propiedades de programas (#property)

PlotIndexSetDouble

Establece el valor de la propiedad correspondiente de línea correspondiente del indicador. La propiedad del indicador tiene que ser del tipo double.

```
bool  PlotIndexSetDouble(
   int     plot_index,     // índice del estilo gráfico
   int     prop_id,        // identificador de la propiedad
   double  prop_value      // valor que se establece
   );
```

Parámetros

plot_index

[in] Índice de la representación gráfica

prop_id

[in] Identificador de la propiedad del indicador. Su valor puede ser uno de los valores de la enumeración ENUM_PLOT_PROPERTY_DOUBLE.

prop_value

[in] Valor de la propiedad.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false.

PlotIndexSetInteger

Establece el valor de la propiedad correspondiente de línea correspondiente del indicador. La propiedad del indicador tiene que ser del tipo int, char, bool o color. Existen 2 variantes de la función.

Llamada con especificación del identificador de la propiedad.

```
bool  PlotIndexSetInteger(
   int  plot_index,         // índice del estilo gráfico
   int  prop_id,            // identificador de la propiedad
   int  prop_value          // valor que se establece
   );
```

Llamada con especificación del identificador y modificador de la propiedad.

```
bool  PlotIndexSetInteger(
   int  plot_index,         // índice del estilo gráfico
   int  prop_id,            // identificador de la propiedad
   int  prop_modifier,      // modificador de la propiedad
   int  prop_value          // valor que se establece
   )
```

Parámetros

plot_index

[in] Índice de la representación gráfica

prop_id

[in] Identificador de la propiedad del indicador. Su valor puede ser uno de los valores de la enumeración ENUM_PLOT_PROPERTY_INTEGER.

prop_modifier

[in] Modificador de la propiedad especificada. Sólo las propiedades de los índices de colores requieren un modificador.

prop_value

[in] Valor de la propiedad.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false.

Ejemplo: indicador que dibuja la línea de tres colores. El esquema de colores se cambia cada 5 ticks.


```
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   1
//---- plot ColorLine
#property indicator_label1  "ColorLine"
#property indicator_type1   DRAW_COLOR_LINE
#property indicator_color1  clrRed,clrGreen,clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  3
//--- indicator buffers
double         ColorLineBuffer[];
double         ColorBuffer[];
int            MA_handle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,ColorLineBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,ColorBuffer,INDICATOR_COLOR_INDEX);
//--- get MA handle
   MA_handle=iMA(Symbol(),0,10,0,MODE_EMA,PRICE_CLOSE);
//---
  }
//+------------------------------------------------------------------+
//| get color index                                                  |
//+------------------------------------------------------------------+
```

```
int getIndexOfColor(int i)
  {
   int j=i%300;
   if(j<100) return(0);// first index
   if(j<200) return(1);// second index
   return(2); // third index
  }
//+------------------------------------------------------------------+
//| 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 ticks=0,modified=0;
   int limit;
//--- first calculation or number of bars was changed
   if(prev_calculated==0)
     {
      //--- copy values of MA into indicator buffer ColorLineBuffer
      int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
      if(copied<=0) return(0);// copying failed - throw away
      //--- now set line color for every bar
      for(int i=0;i<rates_total;i++)
         ColorBuffer[i]=getIndexOfColor(i);
     }
   else
     {
      //--- copy values of MA into indicator buffer ColorLineBuffer
      int copied=CopyBuffer(MA_handle,0,0,rates_total,ColorLineBuffer);
      if(copied<=0) return(0);

      ticks++;// ticks counting
      if(ticks>=5)//it's time to change color scheme
        {
         ticks=0; // reset counter
         modified++; // counter of color changes
         if(modified>=3)modified=0;// reset counter
         ResetLastError();
         switch(modified)
```

```
            {
             case 0:// first color scheme
                PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrRed);
                PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrBlue);
                PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrGreen);
                Print("Color scheme "+modified);
                break;
             case 1:// second color scheme
                PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrYellow);
                PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrPink);
                PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLightSlateGray);
                Print("Color scheme "+modified);
                break;
             default:// third color scheme
                PlotIndexSetInteger(0,PLOT_LINE_COLOR,0,clrLightGoldenrod);
                PlotIndexSetInteger(0,PLOT_LINE_COLOR,1,clrOrchid);
                PlotIndexSetInteger(0,PLOT_LINE_COLOR,2,clrLimeGreen);
                Print("Color scheme "+modified);
            }
        }
      else
        {
         //--- set start position
         limit=prev_calculated-1;
         //--- now we set line color for every bar
         for(int i=limit;i<rates_total;i++)
            ColorBuffer[i]=getIndexOfColor(i);
        }
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

PlotIndexSetString

Establece el valor de la propiedad correspondiente de línea correspondiente del indicador. La propiedad del indicador tiene que ser del tipo string.

```
bool  PlotIndexSetString(
   int     plot_index,     // índice del estilo gráfico
   int     prop_id,        // identificador de la propiedad
   string  prop_value      // valor que se establece
   );
```

Parámetros

plot_index

[in] Índice de la representación gráfica

prop_id

[in] Identificador de la propiedad del indicador. Su valor puede ser uno de los valores de la enumeración ENUM_PLOT_PROPERTY_STRING.

prop_value

[in] Valor de la propiedad.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false.

PlotIndexGetInteger

Devuelve el valor de la propiedad correspondiente de línea correspondiente del indicador. La propiedad del indicador tiene que ser del tipo int, color, bool o char. Existen 2 variantes de la función.

Llamada con especificación del identificador de la propiedad.

```
int  PlotIndexGetInteger(
   int  plot_index,        // índice del estilo gráfico
   int  prop_id,           // identificador de la propiedad
   );
```

Llamada con especificación del identificador y modificador de la propiedad.

```
int  PlotIndexGetInteger(
   int  plot_index,        // índice del estilo gráfico
   int  prop_id,           // identificador de la propiedad
   int  prop_modifier      // modificador de la propiedad
   )
```

Parámetros

plot_index

[in] Índice de construcción gráfica

prop_id

[in] Identificador de la propiedad del indicador. Su valor puede ser uno de los valores de la enumeración ENUM_PLOT_PROPERTY_INTEGER.

prop_modifier

[in] Modificador de la propiedad especificada. Sólo las propiedades de los índices de colores requieren un modificador.

Nota

La función está destinada para extraer las configuraciones de dibujo de una línea correspondiente del indicador. Esta función trabaja junto con la función PlotIndexSetInteger teniendo objetivo de copiar las propiedades de dibujo de una línea en otra.

Ejemplo: El indicador que pinta las velas de un color que depende del día de la semana. Los colores para cada día se establecen de un modo programado.


```
#property indicator_separate_window
#property indicator_buffers 5
#property indicator_plots   1
//---- plot ColorCandles
#property indicator_label1  "ColorCandles"
#property indicator_type1   DRAW_COLOR_CANDLES
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- indicator buffers
double          OpenBuffer[];
double          HighBuffer[];
double          LowBuffer[];
double          CloseBuffer[];
double          ColorCandlesColors[];
color           ColorOfDay[6]={CLR_NONE,clrMediumSlateBlue,
                               clrDarkGoldenrod,clrForestGreen,clrBlueViolet,clrRed};
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
void OnInit()
  {
//--- indicator buffers mapping
   SetIndexBuffer(0,OpenBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,HighBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,LowBuffer,INDICATOR_DATA);
   SetIndexBuffer(3,CloseBuffer,INDICATOR_DATA);
   SetIndexBuffer(4,ColorCandlesColors,INDICATOR_COLOR_INDEX);
//--- set number of colors in color buffer
```

```
   PlotIndexSetInteger(0,PLOT_COLOR_INDEXES,6);
//--- set colors for color buffer
   for(int i=1;i<6;i++)
      PlotIndexSetInteger(0,PLOT_LINE_COLOR,i,ColorOfDay[i]);
//--- set accuracy
   IndicatorSetInteger(INDICATOR_DIGITS,_Digits);
   printf("We have %u colors of days",PlotIndexGetInteger(0,PLOT_COLOR_INDEXES));
//---
  }
//+------------------------------------------------------------------+
//| 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[])
  {
//---
   int i;
   MqlDateTime t;
//----
   if(prev_calculated==0) i=0;
   else i=prev_calculated-1;
//----
   while(i<rates_total)
     {
      OpenBuffer[i]=open[i];
      HighBuffer[i]=high[i];
      LowBuffer[i]=low[i];
      CloseBuffer[i]=close[i];
      //--- set color for every candle
      TimeToStruct(time[i],t);
      ColorCandlesColors[i]=t.day_of_week;
      //---
      i++;
     }
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

Objetos gráficos

Es el grupo de funciones destinadas para trabajar con objetos gráficos pertenecientes a cualquier gráfico especificado. No se puede usar estas funciones en los indicadores.

Función	Acción
ObjectCreate	Crea un objeto del tipo especificado en un gráfico especificado
ObjectName	Devuelve el nombre de un objeto del tipo correspondiente en el gráfico especificado (subventana del gráfico especificada)
ObjectDelete	Elimina el objeto con el nombre especificado del gráfico especificado (subventana del gráfico especificada)
ObjectsDeleteAll	Elimina todos los objetos del tipo especificado del gráfico especificado (subventana del gráfico especificada)
ObjectFind	Busca por el nombre un objeto con el identificador especificado
ObjectGetTimeByValue	Devuelve el valor de hora para el valor especificado del precio de un objeto
ObjectGetValueByTime	Devuelve el valor de precio de un objeto para la hora especificada
ObjectMove	Cambia las coordinadas del punto de anclaje de un objeto
ObjectsTotal	Devuelve el número de objetos del tipo especificado en el gráfico especificado (subventana del gráfico especificada)
ObjectGetDouble	Devuelve el valor del tipo double de la propiedad correspondiente de un objeto
ObjectGetInteger	Devuelve el valor de números enteros de la propiedad correspondiente de un objeto
ObjectGetString	Devuelve el valor del tipo string de la propiedad correspondiente de un objeto
ObjectSetDouble	Establece el valor de propiedad correspondiente de un objeto
ObjectSetInteger	Establece el valor de propiedad correspondiente de un objeto
ObjectSetString	Establece el valor de propiedad correspondiente de un objeto
TextSetFont	Establece la fuente para visualizar el texto a

	través de los métodos del dibujo (por defecto se usa Arial 20)
TextOut	Pasa el texto al array personalizado (búfer) que sirve para crear un recurso gráfico
TextGetSize	Devuelve el alto y el ancho de la cadena con los ajustes de la fuente actuales

Cada objeto gráfico debe tener el nombre único dentro de los márgenes de un gráfico, incluyendo sus subventanas. El cambio del nombre de un objeto gráfico genera dos eventos: evento de eliminación del objeto con el nombre anterior, evento de creación del objeto gráfico con el nombre nuevo.

Después de crear un objeto o modificar las propiedades del objeto se recomienda llamar a la función ChartRedraw(), la que ordena al terminal dibujar el gráfico por vía forzada (y todos sus objetos visibles).

ObjectCreate

Crea un objeto con el nombre especificado, tipo y coordinadas iniciales en la subventana del gráfico especificada. Durante la creación se puede indicar hasta 30 coordinadas.

```
bool  ObjectCreate(
   long          chart_id,      // identificador del gráfico
   string        name,          // nombre del objeto
   ENUM_OBJECT   type,          // tipo de objeto
   int           sub_window,    // índice de ventana
   datetime      time1,         // hora del primer punto de anclaje
   double        price1,        // precio del primer punto de anclaje
   ...
   datetime      timeN=0,       // hora de punto de anclaje N
   double        priceN=0,      // precio de punto de anclaje N
   ...
   datetime      time30=0,      // hora del punto de anclaje 30
   double        price30=0      // precio del punto de anclaje 30
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto. El nombre tiene que ser único dentro de los límites de un gráfico, incluyendo sus subventanas.

type

[in] Tipo de objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT.

sub_window

[in] Número de subventana del gráfico. 0 significa la ventana principal del gráfico. La subventana especificada debe existir, de lo contrario la función devuelve false.

time1

[in] Coordinada de hora del primer enlace.

price1

[in] Coordinada de precio del primer punto de anclaje.

timeN=0

[in] Coordinada de hora del punto de anclaje N.

priceN=0

[in] Coordinada de precio del punto de anclaje N.

time30=0

[in] Coordinada de hora del punto de anclaje 30.

price30=0

[in] Coordinada de precio del punto de anclaje 30.

Valor devuelto

Devuelve true o false dependiendo si el objeto ha sido creado o no. Para la información más detallada sobre el error hay que llamar a la función GetLastError(). Si el objeto ya ha sido creado, se intenta cambiar sus coordinadas.

Nota

La numeración de las subventanas del gráfico (si en el gráfico hay subventanas con indicadores) se empieza desde 1. Siempre existe la venta principal y tiene el índice 0.

La gran cantidad de puntos de anclaje (hasta 30) está prevista para usarlas en el futuro. Al mismo tiempo el límite de 30 posibles puntos de anclaje para los objetos gráficos se debe al hecho que durante la llamada a la función el número de parámetros no debe superar 64.

Al renombrar un objeto gráfico, se generan al mismo tiempo dos eventos que pueden ser procesados en el Asesor Experto o el indicador usando la función OnChartEvent():

- evento de eliminación del objeto con el nombre anterior;
- evento de creación del objeto gráfico con el nombre nuevo.

Para crear cada uno de los siguientes tipos de objetos, es necesario fijar una cierta cantidad de los puntos de anclaje:

Identificador	Descripción	Puntos de anclaje
OBJ_VLINE	Linea vertical	Un punto de anclaje. Prácticamente se utiliza sólo la coordinada del eje de tiempo.
OBJ_HLINE	Linea horizontal	Un punto de anclaje. Prácticamente se utiliza sólo la coordinada del eje de precio.
OBJ_TREND	Línea de tendencia	Dos puntos de anclaje.
OBJ_TRENDBYANGLE	Linea de tendencia por ángulo	Dos puntos de anclaje.
OBJ_CYCLES	Líneas cíclicas	Dos puntos de anclaje.
OBJ_ARROWED_LINE	Objeto "Línea con flecha"	Dos puntos de anclaje.
OBJ_CHANNEL	Canal equidistante	Tres puntos de anclaje.
OBJ_STDDEVCHANNEL	Canal de desviación estándar	Dos puntos de anclaje.
OBJ_REGRESSION	Canal de regresión lineal	Dos puntos de anclaje.
OBJ_PITCHFORK	Tridente Andrews	Tres puntos de anclaje.
OBJ_GANNLINE	Línea de Gann	Dos puntos de anclaje.
OBJ_GANNFAN	Abanico de Gann	Dos puntos de anclaje.
OBJ_GANNGRID	Retícula de Gann	Dos puntos de anclaje.

OBJ_FIBO	Niveles de Fibonacci	Dos puntos de anclaje.
OBJ_FIBOTIMES	Zonas temporales de Fibonacci	Dos puntos de anclaje.
OBJ_FIBOFAN	Abanico de Fibonacci	Dos puntos de anclaje.
OBJ_FIBOARC	Arcos de Fibonacci	Dos puntos de anclaje.
OBJ_FIBOCHANNEL	Canal de Fibonacci	Tres puntos de anclaje.
OBJ_EXPANSION	Expansiones de Fibonacci	Tres puntos de anclaje.
OBJ_ELLIOTWAVE5	5 ondas de Elliott	Cinco puntos de anclaje.
OBJ_ELLIOTWAVE3	3 ondas de Elliott	Tres puntos de anclaje.
OBJ_RECTANGLE	Rectángulo	Dos puntos de anclaje.
OBJ_TRIANGLE	Triángulo	Tres puntos de anclaje.
OBJ_ELLIPSE	Elipse	Tres puntos de anclaje.
OBJ_ARROW_THUMB_UP	Signo "Bien" (pulgar arriba)	Un punto de anclaje.
OBJ_ARROW_THUMB_DOWN	Signo "Mal" (pulgar abajo)	Un punto de anclaje.
OBJ_ARROW_UP	Signo "Flecha arriba"	Un punto de anclaje.
OBJ_ARROW_DOWN	Signo "Flecha abajo"	Un punto de anclaje.
OBJ_ARROW_STOP	Signo "Stop"	Un punto de anclaje.
OBJ_ARROW_CHECK	Signo "Visto" (marca de comprobación)	Un punto de anclaje.
OBJ_ARROW_LEFT_PRICE	Etiqueta izquierda de precio	Un punto de anclaje.
OBJ_ARROW_RIGHT_PRICE	Etiqueta derecha de precio	Un punto de anclaje.
OBJ_ARROW_BUY	Signo "Buy"	Un punto de anclaje.
OBJ_ARROW_SELL	Signo "Sell"	Un punto de anclaje.
OBJ_ARROW	Objeto "Flecha"	Un punto de anclaje.
OBJ_TEXT	Objeto "Texto"	Un punto de anclaje.
OBJ_LABEL	Objeto "Etiqueta de texto"	La posición se fija utilizando las propiedades OBJPROP_XDISTANCE y OBJPROP_YDISTANCE.
OBJ_BUTTON	Objeto "Botón"	La posición se fija utilizando las propiedades OBJPROP_XDISTANCE y OBJPROP_YDISTANCE.
OBJ_CHART	Objeto "Gráfico"	La posición se fija utilizando las propiedades OBJPROP_XDISTANCE y

		OBJPROP_YDISTANCE.
OBJ_BITMAP	Objeto "Imagen"	Un punto de anclaje.
OBJ_BITMAP_LABEL	Objeto "Etiqueta gráfica"	La posición se fija utilizando las propiedades OBJPROP_XDISTANCE y OBJPROP_YDISTANCE.
OBJ_EDIT	Objeto "Campo de edición"	La posición se fija utilizando las propiedades OBJPROP_XDISTANCE y OBJPROP_YDISTANCE.
OBJ_EVENT	Objeto "Evento" que corresponde a un evento en el calendario económico	Un punto de anclaje. Prácticamente se utiliza sólo la coordinada del eje de tiempo.
OBJ_RECTANGLE_LABEL	Objeto "Etiqueta rectangular" para crear y personalizar la interfaz gráfica de usuario.	La posición se fija utilizando las propiedades OBJPROP_XDISTANCE y OBJPROP_YDISTANCE.

ObjectName

Devuelve el nombre del objeto correspondiente en el gráfico especificado (subventana del gráfico especificada) del tipo especificado.

```
string  ObjectName(
   long  chart_id,           // identificador del gráfico
   int   pos,                // número en la lista de objetos
   int   sub_window=-1,      // número de ventana
   int   type=-1             // tipo de objeto
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

pos

[in] Número ordinal del objeto de acuerdo con el filtro especificado según el número de subventana y tipo.

nwin=-1

[in] Número de subventana del gráfico. 0 significa la ventana principal, -1 significa todas las subventanas del gráfico, incluyendo la ventana principal.

type=-1

[in] Tipo del objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT. -1 significa todos los tipos.

Valor devuelto

Nombre del objeto en caso del éxito.

Nota

Al renombrar un objeto gráfico, se generan al mismo tiempo dos eventos que pueden ser procesados en el Asesor Experto o el indicador usando la función OnChartEvent():

- evento de eliminación del objeto con el nombre anterior;
- evento de creación del objeto gráfico con el nombre nuevo.

ObjectDelete

Elimina el objeto con el nombre especificado del gráfico especificado.

```
bool  ObjectDelete(
   long    chart_id,      // identificador del gráfico
   string  name           // nombre del objeto
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto a eliminar.

Valor devuelto

Devuelve true en caso de la eliminación con éxito, de lo contrario devuelve false. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

Nota

Al renombrar un objeto gráfico, se generan al mismo tiempo dos eventos que pueden ser procesados en el Asesor Experto o el indicador usando la función OnChartEvent():

- evento de eliminación del objeto con el nombre anterior;
- evento de creación del objeto gráfico con el nombre nuevo.

ObjectsDeleteAll

Elimina todos los objetos del tipo especificado del gráfico especificado (subventana del gráfico especificada).

```
int  ObjectsDeleteAll(
   long  chart_id,           // identificador del gráfico
   int   sub_window=-1,      // índice de ventana
   int   type=-1             // tipo del objeto a eliminar
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

sub_window=-1

[in] Número de subventana del gráfico. 0 significa la ventana principal, -1 significa todas las subventanas del gráfico, incluyendo la ventana principal.

type=-1

[in] Tipo del objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT. -1 significa todos los tipos.

Valor devuelto

Devuelve el número de objetos eliminados. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

ObjectFind

Busca un objeto con el nombre especificado en el gráfico con el identificador especificado.

```
int  ObjectFind(
   long    chart_id,      // identificador del gráfico
   string  name           // nombre del objeto
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto que se busca.

Valor devuelto

En caso del éxito la función devuelve el número de subventana (0 significa la ventana principal del gráfico) donde se encuentra el objeto encontrado. Si el objeto no ha sido encontrado, la función devuelve un número negativo. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

Nota

Al renombrar un objeto gráfico, se generan al mismo tiempo dos eventos que pueden ser procesados en el Asesor Experto o el indicador usando la función OnChartEvent():

- evento de eliminación del objeto con el nombre anterior;
- evento de creación del objeto gráfico con el nombre nuevo.

ObjectGetTimeByValue

Devuelve el valor de la hora para el valor especificado del precio de un objeto especificado.

```
datetime  ObjectGetTimeByValue(
   long     chart_id,      // identificador del gráfico
   string   name,          // nombre del objeto
   double   value,         // precio
   int      line_id        // línea
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto.

value

[in] Valor del precio.

line_id

[in] Identificador de la línea.

Valor devuelto

Devuelve el valor de la hora para el valor especificado del precio de un objeto especificado.

Nota

Puesto que el objeto puede tener varios valores en una coordinada de precio, en este caso es necesario especificar el indicador de la línea. Esta función se puede aplicar sólo a los siguientes objetos:

- Línea de tendencia (OBJ_TREND)
- Línea de tendencia por ángulo (OBJ_TRENDBYANGLE)
- Línea de Gann (OBJ_GANNLINE)
- Canal equidistante (OBJ_CHANNEL) - 2 líneas
- Canal de regresión lineal (OBJ_REGRESSION) - 3 líneas
- Canal de desviación estándar (OBJ_STDDEVCHANNEL) - 3 líneas
- Flecha (OBJ_ARROWED_LINE)

ObjectGetValueByTime

Devuelve el valor de precio para la hora especificada de un objeto especificado.

```
double  ObjectGetValueByTime(
   long      chart_id,     // identificador del gráfico
   string    name,         // nombre del objeto
   datetime  time,         // hora
   int       line_id       // línea
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto.

time

[in] Valor de la hora.

line_id

[in] Identificador de la línea.

Valor devuelto

Devuelve el valor de precio para la hora especificada de un objeto especificado.

Nota

Puesto que el objeto puede tener varios valores en una coordinada de precio, en este caso es necesario especificar el indicador de la línea. Esta función se puede aplicar sólo a los siguientes objetos:

- Línea de tendencia (OBJ_TREND)
- Línea de tendencia por ángulo (OBJ_TRENDBYANGLE)
- Línea de Gann (OBJ_GANNLINE)
- Canal equidistante (OBJ_CHANNEL) - 2 líneas
- Canal de regresión lineal (OBJ_REGRESSION) - 3 líneas
- Canal de desviación estándar (OBJ_STDDEVCHANNEL) - 3 líneas
- Flecha (OBJ_ARROWED_LINE)

ObjectMove

Cambia las coordinadas del punto de anclaje de un objeto.

```
bool  ObjectMove(
   long      chart_id,        // identificador del gráfico
   string    name,            // nombre del objeto
   int       point_index,     // número de anclaje
   datetime  time,            // hora
   double    price            // precio
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto.

point_index

[in] Número del punto de anclaje. La cantidad de puntos de anclaje depende del tipo de objeto.

time

[in] Coordinada de hora del punto de anclaje especificado.

price

[in] Coordinada de precio del punto de anclaje especificado.

Valor devuelto

En caso del éxito devuelve true, en caso del fallo devuelve false. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

ObjectsTotal

Devuelve el número de objetos del tipo especificado en el gráfico especificado (subventana del gráfico especificada).

```
int  ObjectsTotal(
   long  chart_id,           // identificador del gráfico
   int   sub_window=-1,      // índice de ventana
   int   type=-1             // tipo de objeto
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

nwin=-1

[in] Número de subventana del gráfico. 0 significa la ventana principal, -1 significa todas las subventanas del gráfico, incluyendo la ventana principal.

type=-1

[in] Tipo del objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT. -1 significa todos los tipos.

Valor devuelto

Número de objetos.

ObjectSetDouble

Establece el valor de la propiedad correspondiente de un objeto. La propiedad del objeto debe ser del tipo double. Existen 2 variantes de la función.

Configuración del valor de la propiedad sin modificador

```
bool  ObjectSetDouble(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre
   int      prop_id,           // propiedad
   double   prop_value         // valor
   );
```

Configuración del valor de la propiedad indicando el modificador

```
bool  ObjectSetDouble(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre
   int      prop_id,           // propiedad
   int      prop_modifier,     // modificador
   double   prop_value         // valor
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto.

prop_id

[in] Identificador de la propiedad del objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT_PROPERTY_DOUBLE.

prop_modifier

[in] Modificador de la propiedad especificada. La mayoría de las propiedades no requiere un modificador.

prop_value

[in] Valor de la propiedad.

Valor devuelto

Devuelve true sólo si el comando de modificar las propiedades de un objeto gráfico se ha enviado con éxito al gráfico. De lo contrario devuelve false. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

Ejemplo de creación de un objeto Fibonacci y adicción de un nuevo nivel en él

```
//| Script program start function                                    |
//+------------------------------------------------------------------+
```

```
void OnStart()
  {
//--- matrices auxiliares
   double high[],low[],price1,price2;
   datetime time[],time1,time2;
//--- vamos a copiar los precios de apertura - 100 últimas barras será suficiente
   int copied=CopyHigh(Symbol(),0,0,100,high);
   if(copied<=0)
     {
      Print("Fallo al copiar los valores de la serie de precios High");
      return;
     }
//--- vamos a copiar los precios de cierre - 100 últimas barras será suficiente
   copied=CopyLow(Symbol(),0,0,100,low);
   if(copied<=0)
     {
      Print("Fallo al copiar los valores de la serie de precios Low");
      return;
     }
//--- vamos a copiar la hora de apertura para 100 últimas barras
   copied=CopyTime(Symbol(),0,0,100,time);
   if(copied<=0)
     {
      Print("Fallo al copiar los valores de la serie de precios Time");
      return;
     }
//--- vamos a organizar el acceso a los datos copiados como en las series temporales
   ArraySetAsSeries(high,true);
   ArraySetAsSeries(low,true);
   ArraySetAsSeries(time,true);

//--- coordinadas del primer punto de anclaje del objeto Fibo
   price1=high[70];
   time1=time[70];
//--- coordinadas del segundo punto de anclaje del objeto Fibo
   price2=low[50];
   time2=time[50];

//--- ya es hora de crear el mismo objeto Fibo
   bool created=ObjectCreate(0,"Fibo",OBJ_FIBO,0,time1,price1,time2,price2);
   if(created) // si el objeto ha sido creado con éxito
     {
      //--- vamos a configurar los colores de niveles Fibo
      ObjectSetInteger(0,"Fibo",OBJPROP_LEVELCOLOR,Blue);
      //--- por cierto, ¿cuántos niveles Fibo tenemos nosotros
      int levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
      Print("Fibo levels before = ",levels);
      //---mostramos en el Diario => número del nivel:valor de descripción_de_nivel
```

```
      for(int i=0;i<levels;i++)
        {
         Print(i,":",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
               "  ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
        }
      //--- vamos a intentar aumentar la cantidad de niveles a uno
      bool modified=ObjectSetInteger(0,"Fibo",OBJPROP_LEVELS,levels+1);
      if(!modified) // no ha salido cambiar la cantidad de niveles
        {
         Print("Fallo al cambiar el número de niveles de Fibo, error ",GetLastError()
        }
      //--- simplemente avisamos
      Print("Fibo levels after =",ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS));
      //--- establecemos el valor para el nivel recién creado
      bool added=ObjectSetDouble(0,"Fibo",OBJPROP_LEVELVALUE,levels,133);
      if(added) // hemos podido establecer el valor para este nivel
        {
         Print("Hemos establecido con éxito otro nivel más de Fibo");
         //--- también hay que establecer la descripción del nivel
         ObjectSetString(0,"Fibo",OBJPROP_LEVELTEXT,levels,"my level");
         ChartRedraw(0);
         //--- obtenemos el valor actual de la cantidad de niveles en el objeto Fibo
         levels=ObjectGetInteger(0,"Fibo",OBJPROP_LEVELS);
         Print("Fibo levels after adding = ",levels);
         //--- otra vez visualizamos todos los niveles, simplemente para comprobar
         for(int i=0;i<levels;i++)
           {
            Print(i,":",ObjectGetDouble(0,"Fibo",OBJPROP_LEVELVALUE,i),
                  "  ",ObjectGetString(0,"Fibo",OBJPROP_LEVELTEXT,i));
           }

        }
      else // fallo al intentar aumentar la cantidad de niveles en el objeto Fibo
        {
         Print("Fallo al establecer otro nivel más de Fibo. Error",GetLastError());
        }
     }
  }
```

Véase también

Tipos de objetos, Propiedades de objetos

ObjectSetInteger

Establece el valor de la propiedad correspondiente de un objeto. La propiedad del objeto debe ser de los tipos datetime, int, color, bool o char. Existen 2 variantes de la función.

Configuración del valor de la propiedad sin modificador

```
bool  ObjectSetInteger(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre
   int      prop_id,           // propiedad
   long     prop_value         // valor
   );
```

Configuración del valor de la propiedad indicando el modificador

```
bool  ObjectSetInteger(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre
   int      prop_id,           // propiedad
   int      prop_modifier,     // modificador
   long     prop_value         // valor
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto.

prop_id

[in] Identificador de la propiedad del objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT_PROPERTY_INTEGER.

prop_modifier

[in] Modificador de la propiedad especificada. La mayoría de las propiedades no requiere un modificador.

prop_value

[in] Valor de la propiedad.

Valor devuelto

Devuelve true sólo si el comando de modificar las propiedades de un objeto gráfico se ha enviado con éxito al gráfico. De lo contrario devuelve false. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

Un ejemplo de cómo crear una tabla de colores Web

```
//+------------------------------------------------------------------+
//|                                                Table of Web Colors|
//|                        Copyright 2011, MetaQuotes Software Corp |
//|                                             http://www.metaquotes.net |
//+------------------------------------------------------------------+
#define X_SIZE 140      // ancho del objeto OBJ_EDIT
#define Y_SIZE 33       // ancho del objeto OBJ_EDIT
//+------------------------------------------------------------------+
//| Array de colores Web
//+------------------------------------------------------------------+
color ExtClr[140]=
  {
   clrAliceBlue,clrAntiqueWhite,clrAqua,clrAquamarine,clrAzure,clrBeige,clrBisque,clr
   clrBlue,clrBlueViolet,clrBrown,clrBurlyWood,clrCadetBlue,clrChartreuse,clrChocolat
   clrCornsilk,clrCrimson,clrCyan,clrDarkBlue,clrDarkCyan,clrDarkGoldenrod,clrDarkGra
   clrDarkMagenta,clrDarkOliveGreen,clrDarkOrange,clrDarkOrchid,clrDarkRed,clrDarkSal
   clrDarkSlateBlue,clrDarkSlateGray,clrDarkTurquoise,clrDarkViolet,clrDeepPink,clrDe
   clrDodgerBlue,clrFireBrick,clrFloralWhite,clrForestGreen,clrFuchsia,clrGainsboro,c
   clrGoldenrod,clrGray,clrGreen,clrGreenYellow,clrHoneydew,clrHotPink,clrIndianRed,c
   clrLavender,clrLavenderBlush,clrLawnGreen,clrLemonChiffon,clrLightBlue,clrLightCor
   clrLightGoldenrod,clrLightGreen,clrLightGray,clrLightPink,clrLightSalmon,clrLightS
   clrLightSlateGray,clrLightSteelBlue,clrLightYellow,clrLime,clrLimeGreen,clrLinen,c
   clrMediumAquamarine,clrMediumBlue,clrMediumOrchid,clrMediumPurple,clrMediumSeaGree
   clrMediumSpringGreen,clrMediumTurquoise,clrMediumVioletRed,clrMidnightBlue,clrMint
   clrNavajoWhite,clrNavy,clrOldLace,clrOlive,clrOliveDrab,clrOrange,clrOrangeRed,clr
   clrPaleGreen,clrPaleTurquoise,clrPaleVioletRed,clrPapayaWhip,clrPeachPuff,clrPeru,
   clrPurple,clrRed,clrRosyBrown,clrRoyalBlue,clrSaddleBrown,clrSalmon,clrSandyBrown,
   clrSienna,clrSilver,clrSkyBlue,clrSlateBlue,clrSlateGray,clrSnow,clrSpringGreen,cl
   clrThistle,clrTomato,clrTurquoise,clrViolet,clrWheat,clrWhite,clrWhiteSmoke,clrYel
  };
//+------------------------------------------------------------------+
//| Creación e inicialización del objeto OBJ_EDIT                    |
//+------------------------------------------------------------------+
void CreateColorBox(int x,int y,color c)
  {
//--- generaremos el nombre para nuevo objeto según el nombre del color
   string name="ColorBox_"+(string)x+"_"+(string)y;
//--- crearemos nuevo objeto OBJ_EDIT
   if(!ObjectCreate(0,name,OBJ_EDIT,0,0,0))
     {
      Print("Cannot create: '",name,"'");
      return;
     }
//--- fijaremos las coordinadas del punto de enlace, ancho y alto en píxeles
   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);
//--- estableceremos el color del texto del objeto
   if(clrBlack==c) ObjectSetInteger(0,name,OBJPROP_COLOR,clrWhite);
   else            ObjectSetInteger(0,name,OBJPROP_COLOR,clrBlack);
//--- estableceremos el color del fondo
   ObjectSetInteger(0,name,OBJPROP_BGCOLOR,c);
//--- estableceremos el texto del objeto OBJ_EDIT correspondiente al color del fondo
   ObjectSetString(0,name,OBJPROP_TEXT,(string)c);
  }
//+------------------------------------------------------------------+
//| Función del inicio del script                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
```

```
//--- crearemos una tabla de los bloques coloreados 7x20
   for(uint i=0;i<140;i++)
      CreateColorBox(i%7,i/7,ExtClr[i]);
  }
```

Véase también

Tipos de objetos, Propiedades de objetos

ObjectSetString

Establece el valor de la propiedad correspondiente de un objeto. La propiedad del objeto debe ser del tipo string. Existen 2 variantes de la función.

Configuración del valor de la propiedad sin modificador

```
bool  ObjectSetString(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre
   int      prop_id,           // propiedad
   string   prop_value         // valor
   );
```

Configuración del valor de la propiedad indicando el modificador

```
bool  ObjectSetString(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre
   int      prop_id,           // propiedad
   int      prop_modifier,     // modificador
   string   prop_value         // valor
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto.

prop_id

[in] Identificador de la propiedad del objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT_PROPERTY_STRING.

prop_modifier

[in] Modificador de la propiedad especificada. La mayoría de las propiedades no requiere un modificador.

prop_value

[in] Valor de la propiedad.

Valor devuelto

Devuelve true sólo si el comando de modificar las propiedades de un objeto gráfico se ha enviado con éxito al gráfico. De lo contrario devuelve false. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

Nota

Al renombrar un objeto gráfico, se generan al mismo tiempo dos eventos que pueden ser procesados en el Asesor Experto o el indicador usando la función OnChartEvent():

- evento de eliminación del objeto con el nombre anterior;
- evento de creación del objeto gráfico con el nombre nuevo.

ObjectGetDouble

Devuelve el valor de la correspondiente propiedad del objeto. La propiedad del objeto debe ser del tipo double. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
double  ObjectGetDouble(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre del objeto
   int      prop_id,           // identificador de la propiedad
   int      prop_modifier=0    // modificador de la propiedad, si hace falta
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso de éxito el valor de la propiedad se coloca en una variable receptora que se pasa por referencia por el último parámetro.

```
bool  ObjectGetDouble(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre del objeto
   int      prop_id,           // identificador de la propiedad
   int      prop_modifier,     // modificador de la propiedad
   double&  double_var         // aquí recibimos el valor de la propiedad
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto.

prop_id

[in] Identificador de la propiedad del objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT_PROPERTY_DOUBLE.

prop_modifier

[in] Modificador de la propiedad especificada. Para la primera opción el valor del modificador es igual a 0 por defecto. La mayoría de las propiedades no requiere un modificador.

double_var

[out] Variable del tipo double que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo double para la primera opción de la llamada.

Para la segunda variante devuelve true, si dicha propiedad se mantiene y su valor ha sido colocado en la variable double_var, de lo contrario devuelve false. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

ObjectGetInteger

Devuelve el valor de la correspondiente propiedad del objeto. La propiedad del objeto debe ser de los tipos datetime, int, color, bool o char. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
long  ObjectGetInteger(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre del objeto
   int      prop_id,           // identificador de la propiedad
   int      prop_modifier=0    // modificador de la propiedad, si hace falta
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso de éxito el valor de la propiedad se coloca en una variable receptora que se pasa por referencia por el último parámetro.

```
bool  ObjectGetInteger(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre del objeto
   int      prop_id,           // identificador de la propiedad
   int      prop_modifier,     // modificador de la propiedad
   long&    long_var           // aquí recibimos el valor de la propiedad
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto.

prop_id

[in] Identificador de la propiedad del objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT_PROPERTY_INTEGER.

prop_modifier

[in] Modificador de la propiedad especificada. Para la primera opción el valor del modificador es igual a 0 por defecto. La mayoría de las propiedades no requiere un modificador.

long_var

[out] Variable del tipo long que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo long para la primera opción de la llamada.

Para la segunda variante devuelve true, si dicha propiedad se mantiene y su valor ha sido colocado en la variable long_var, de lo contrario devuelve false. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

ObjectGetString

Devuelve el valor de la correspondiente propiedad del objeto. La propiedad del objeto debe ser del tipo string. Existen 2 variantes de la función.

1. Devuelve el valor de la propiedad directamente.

```
string  ObjectGetString(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre del objeto
   int      prop_id,           // identificador de la propiedad
   int      prop_modifier=0    // modificador de la propiedad, si hace falta
   );
```

2. Devuelve true o false dependiendo del éxito de ejecución de la función. En caso de éxito el valor de la propiedad se coloca en una variable receptora que se pasa por referencia por el último parámetro.

```
bool  ObjectGetString(
   long     chart_id,          // identificador del gráfico
   string   name,              // nombre del objeto
   int      prop_id,           // identificador de la propiedad
   int      prop_modifier,     // modificador de la propiedad
   string&  string_var         // aquí recibimos el valor de la propiedad
   );
```

Parámetros

chart_id

[in] Identificador del gráfico. 0 significa el gráfico actual.

name

[in] Nombre del objeto.

prop_id

[in] Identificador de la propiedad del objeto. Su valor puede ser uno de los valores de la enumeración ENUM_OBJECT_PROPERTY_STRING.

prop_modifier

[in] Modificador de la propiedad especificada. Para la primera opción el valor del modificador es igual a 0 por defecto. La mayoría de las propiedades no requiere un modificador.

string_var

[out] Variable del tipo string que recibe el valor de la propiedad requerida.

Valor devuelto

Valor del tipo string para la primera opción de la llamada.

Para la segunda variante devuelve true, si dicha propiedad se mantiene y su valor ha sido colocado en la variable string_var, de lo contrario devuelve false. Para la información más detallada sobre el error hay que llamar a la función GetLastError().

Nota

Al renombrar un objeto gráfico, se generan al mismo tiempo dos eventos que pueden ser procesados en el Asesor Experto o el indicador usando la función OnChartEvent():

- evento de eliminación del objeto con el nombre anterior;
- evento de creación del objeto gráfico con el nombre nuevo.

TextSetFont

Establece la fuente para visualizar el texto a través de los métodos del dibujo y devuelve el resultado de esta operación. Por defecto se usa la fuente Arial con el tamaño -120 (12 pt).

```
bool  TextSetFont(
   const string  name,            // nombre de la fuente o la ruta del archivo de la
   int           size,            // tamaño de la fuente
   uint          flags,           // combinación de banderas
   int           orientation=0    // ángulo de inclinación del texto
   );
```

Parámetros

name

[in] Nombre de la fuente en el sistema, o el nombre del recurso que contiene la fuente, o la ruta del archivo de la fuente en el disco.

size

[in] Tamaño de la fuente que puede ser establecido con valores positivos y negativos. En caso de valores positivos, el tamaño del texto a mostrar no depende de los ajustes de tamaños de las fuentes en el sistema operativo. En caso de valores negativos, el valor se establece en las décimas partes del punto y el tamaño del texto va a depender de los ajustes del sistema ("escala estándar" o "escala grande"). Más detalles sobre la diferencia entre los regímenes se puede leer en la Nota.

flags

[in] Combinación de banderas que describen el estilo de la fuente.

orientation

[in] Ángulo de inclinación del texto por la horizontal respecto al eje X, la unidad de medición es de 0,1 del grado. Es decir, *orientation=450* significa la inclinación bajo un ángulo de 45 grados.

Valor devuelto

Devuelve true si la fuente actual se establece con éxito, de lo contrario es false. Posibles códigos de errores:

- ERR_INVALID_PARAMETER(4003) - *name* representa NULL o "" (cadena vacía),
- ERR_INTERNAL_ERROR(4001) - error del sistema operativo (por ejemplo, intento de crear una fuente que no existe).

Nota

Si en el nombre de la fuente se utiliza "::", la fuente se carga desde un recurso EX5. Si el nombre de la fuente *name* va acompañado con la extensión, entonces esta fuente se carga desde el archivo. En esta caso, si la ruta de la fuente se empieza con "\" o "/", el archivo se busca referente la carpeta MQL5, de lo contrario se busca referente a la ruta del archivo EX5 que ha llamado a la función TextSetFont().

El tamaño de la fuente se establece con valores positivos o negativos, el signo determina la dependencia del tamaño de la fuente de los ajustes del sistema operativo (la escala de la fuente).

- Si para establecer el tamaño se utiliza un número positivo, este tamaño se transforma en unidades

físicas de medición del dispositivo (píxeles) durante el cambio de la fuente lógica a la física, y este tamaño corresponde a la altura de los glifos del símbolo escogidos desde las fuentes disponibles. Este caso no es recomendable cuando se supone el uso común de los textos visualizados por la función TextOut() y los textos visualizados a través del objeto gráfico OBJ_LABEL ("Etiqueta de texto") en el mismo gráfico.

- Si para establecer el tamaño se utiliza un número negativo, se supone que el número especificado se establece en décimas partes del punto lógico (el valor -350 es igual a 35 puntos lógicos) y se divide por 10. Luego el valor obtenido se transforma en unidades físicas de medición del dispositivo (píxeles) y corresponde al valor absoluto de la altura del símbolo desde las fuentes disponibles. Para conseguir en la pantalla el texto con el mismo tamaño que hay en el objeto OBJ_LABEL, es necesario multiplicar el tamaño de la fuente especificada en las propiedades del objeto por -10.

Las banderas pueden ser utilizadas como combinaciones de las banderas del estilo con una de las banderas que determina el grosor de la fuente. Los nombres de las banderas se listan a continuación.

Banderas para fijar el estilo de escritura de la fuente

Bandera	Descripción
FONT_ITALIC	Cursiva
FONT_UNDERLINE	Subrayado
FONT_STRIKEOUT	Tachado

Banderas para fijar el grosor de la fuente

Bandera
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

Véase también

Recursos, ResourceCreate(), ResourceSave(), TextOut()

TextOut

Esta función visualiza el texto en un array personalizado (búfer) y devuelve el resultado de esta operación. Este array está destinado para la generación de un recurso gráfico.

```
bool  TextOut(
   const string       text,          // texto mostrado
   int                x,             // coordenada X
   int                y,             // coordenada Y
   uint               anchor,        // modo de anclaje
   uint               &data[],       // búfer de salida
   uint               width,         // ancho del búfer en píxeles
   uint               height,        // alto del búfer en píxeles
   uint               color,         // color del texto
   ENUM_COLOR_FORMAT  color_format   // formato del texto a visualizar
   );
```

Parámetros

text

[in] Texto visualizado que va a ser escrito en el búfer. Se visualiza sólo el texto de una solo línea.

x

[in] Coordenada X del punto de anclaje para el texto visualizado.

y

[in] Coordenada Y del punto de anclaje para el texto visualizado.

anchor

[in] Valor del conjunto de 9 modos predeterminados de la posición del punto de anclaje del texto visualizado. La posición se establece con la combinación de dos banderas: la bandera de alineación del texto por la horizontal y la bandera de alineación del texto por la vertical. Los nombres de las banderas se listan en la Nota de abajo.

data[]

[in] Búfer que recibe el texto. Este búfer se utiliza para crear un recurso gráfico.

width

[in] Ancho del búfer en puntos (píxeles).

height

[in] Alto del búfer en puntos (píxeles).

color

[in] Color del texto.

color_format

[in] Formato del color se establece con un valor desde la enumeración ENUM_COLOR_FORMAT.

Valor devuelto

Devuelve true en caso de la ejecución exitosa, de lo contrario devuelve false.

Nota

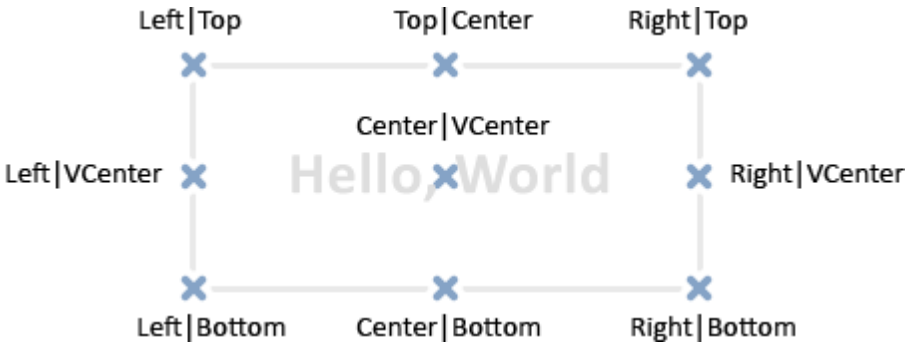
El modo de anclaje que se establece con el parámetro `anchor` es una combinación de dos banderas de alineación de texto por la vertical y por la horizontal. Banderas de alineación de texto por la horizontal:

- TA_LEFT - punto de anclaje situado en la parte izquierda del rectángulo limitador
- TA_CENTER - punto de anclaje por la horizontal se sitúa en el centro del rectángulo limitador
- TA_RIGHT - punto de anclaje situado en la parte derecha del rectángulo limitador

Banderas de alineación de texto por la vertical

- TA_TOP - punto de anclaje situado en la parte superior del rectángulo limitador
- TA_VCENTER - punto de anclaje por la vertical se sitúa en el centro del rectángulo limitador
- TA_BOTTOM - punto de anclaje situado en la parte inferior del rectángulo limitador

Las posibles combinaciones de banderas y los modos de anclaje que éstas establecen se muestran en la figura de abajo.



Ejemplo:

```
//--- ancho y alto del canvas (lienzo en el que se hace el dibujo)
#define IMG_WIDTH  200
#define IMG_HEIGHT 200
//--- antes de iniciar el script mostramos la ventana con los parámetros
#property script_show_inputs
//--- damos la posibilidad de establecer el formato del color
input ENUM_COLOR_FORMAT clr_format=COLOR_FORMAT_XRGB_NOALPHA;
//--- array (búfer) de dibujar
uint ExtImg[IMG_WIDTH*IMG_HEIGHT];
//+------------------------------------------------------------------+
//| Script program start function                                    |
//+------------------------------------------------------------------+
void OnStart()
  {
//--- creamos el objeto OBJ_BITMAP_LABEL para dibujar
   ObjectCreate(0,"CLOCK",OBJ_BITMAP_LABEL,0,0,0);
//--- especificamos el nombre del recurso gráfico para escribir en el objeto CLOCK
   ObjectSetString(0,"CLOCK",OBJPROP_BMPFILE,"::IMG");

//--- variables auxiliares
   double a;            // ángulo de la flecha
   uint   nm=2700;      // contador de minutos
   uint   nh=2700*12;   // contador de horas
   uint   w,h;          // variables para obtener los tamaños de las cadenas de texto
   int    x,y;          // variables para calcular las coordenadas actuales del punto

//--- rotamos las agujas del reloj en un ciclo infinito hasta la detención del script
   while(!IsStopped())
     {
      //--- limpiamos el array del búfer para el dibujo del reloj
      ArrayFill(ExtImg,0,IMG_WIDTH*IMG_HEIGHT,0);
      //--- establecer la fuente para dibujar las cifras sobre la carátula del reloj
      TextSetFont("Courier",30,FW_EXTRABOLD,0);
      //--- dibujamos la carátula del reloj
      for(int i=1;i<=12;i++)
        {
         //--- obtenemos el tamaño de la hora actual sobre la carátula
         TextGetSize(string(i),w,h);
         //--- calculamos coordenadas de la hora actual sobre la carátula
         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);
         //--- visualización de esta hora sobre la carátula en el búfer ExtImg[]
         TextOut(string(i),x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,
        }
      //--- ahora establecemos la fuente para dibujar el minutero
      TextSetFont("Courier",20,FW_EXTRABOLD,(uint)-(nm%3600));
      //--- obtenemos el tamaño del minutero
      TextGetSize("---->",w,h);
      //--- calculamos las coordenadas del minutero sobre la carátula del reloj
      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);
      //--- visualización del minutero sobre la carátula en el búfer ExtImg[]
      TextOut("---->",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_f

      //--- ahora establecemos la fuente para dibujar el horario
      TextSetFont("Courier",20,FW_EXTRABOLD,(uint)-(nh/12%3600));
      TextGetSize("===>",w,h);
      //--- calculamos las coordenadas del horario sobre la carátula del reloj
      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);
      //--- visualización del horario sobre la carátula en el búfer ExtImg[]
      TextOut("===>",x,y,TA_LEFT|TA_TOP,ExtImg,IMG_WIDTH,IMG_HEIGHT,0xFFFFFFFF,clr_fo

      //--- actualización del recurso gráfico
      ResourceCreate("::IMG",ExtImg,IMG_WIDTH,IMG_HEIGHT,0,0,IMG_WIDTH,clr_format);
      //--- actualización forzada del gráfico
      ChartRedraw();

      //--- aumentamos los contadores de hora y minutos
      nm+=60;
      nh+=60;
      //--- mantenemos una pausa entre los cuadros
      Sleep(100);
     }
//--- eliminamos el objeto CLOCK al terminarse el trabajo del script
   ObjectDelete(0,"CLOCK");
//---
  }
```

Véase también

Recursos, ResourceCreate(), ResourceSave(), TextGetSize(), TextSetFont()

TextGetSize

Devuelve el alto y el ancho de la cadena con actuales ajustes de la fuente.

```
bool  TextGetSize(
   const string       text,          // cadena del texto
   uint&               width,        // ancho del búfer en píxeles
   uint&               height        // alto del búfer en píxeles
   );
```

Parámetros

text

[in] Cadena para la que obtenemos el largo y el ancho.

width

[out] Parámetro de entrada para obtener el ancho.

height

[out] Parámetro de entrada para obtener el alto.

Valor devuelto

Devuelve true en caso de la ejecución exitosa, de lo contrario devuelve false. Posibles códigos de errores:

- ERR_INTERNAL_ERROR(4001) - en caso del error del sistema operativo.

Véase también

Recursos, ResourceCreate(), ResourceSave(), TextSetFont(), TextOut()

Funciones para trabajar con indicadores técnicos

Todas las funciones del tipo iMA, iAC, iMACD, iIchimoku etc. crean una copia del indicador técnico correspondiente en la caché global del terminal de cliente. Si la copia del indicador con estos parámetros ya existe, una copia nueva no se crea, sólo el contador de referencias a esta copia existente se aumenta.

Estas funciones devuelven el manejador (handle) de la copia del indicador correspondiente. En el futuro, si usamos este manejador, se puede recibir los datos calculados por el indicador correspondiente. Los datos del buffer correspondiente (los indicadores técnicos contienen los datos calculados en sus buffers internos el número de los cuales puede ser de 1 a 5, dependiendo de cada indicador) pueden ser copiados en un programa mql5 usando la función CopyBuffer().

No se puede acudir a los datos del indicador justo después de su creación porque el cálculo de valores de indicador requiere un tiempo, por eso es mejor crear los manejadores de indicadores en OnInit(). La función iCustom() crea el indicador personalizado correspondiente y devuelve su manejador en caso de crearlo con éxito. Los indicadores personalizados pueden contener de hasta 512 buffers de indicador, el contenido de los cuales también se puede obtener mediante la función CopyBuffer(), usando el manejador obtenido.

Existe un método universal para crear cualquier indicador técnico con la función IndicatorCreate(). Esta función acepta los siguientes parámetros de entrada:

- nombre del símbolo;
- período de tiempo;
- tipo del indicador que se crea;
- número de parámetros de entrada del indicador;
- array del tipo MqlParam que contiene todos los parámetros de entrada necesarios.

Para liberar la memoria del ordenador de un indicador que ya no se usa existe la función IndicatorRelease() a la que se pasa el manejador de este indicador.

Nota. Llamada repetida a la función del indicador con los mismos parámetros dentro de un programa mql5 no llevará al aumento repetido del contador de referencias; el contador será aumentado sólo una vez a 1. Sin embargo, se recomienda obtener los manejadores de indicadores en la función OnInit() o en el constructor de clase, con su posterior uso en las demás funciones. El contador de referencias se disminuye cuando un programa mql5 se deinicializa.

Todas las funciones de indicador disponen como mínimo de 2 parámetros: símbolo y período. El valor del símbolo NULL significa el instrumento actual, el valor del período 0 significa el período de tiempo corriente.

Función	Devuelve el manejador del indicador:
iAC	Accelerator Oscillator
iAD	Accumulation/Distribution
iADX	Average Directional Index
iADXWilder	Average Directional Index by Welles Wilder
iAlligator	Alligator

iAMA	Adaptive Moving Average
iAO	Awesome Oscillator
iATR	Average True Range
iBearsPower	Bears Power
iBands	Bollinger Bands®
iBullsPower	Bulls Power
iCCI	Commodity Channel Index
iChaikin	Chaikin Oscillator
iCustom	indicador personalizado
iDEMA	Double Exponential Moving Average
iDeMarker	DeMarker
iEnvelopes	Envelopes
iForce	Force Index
iFractals	Fractals
iFrAMA	Fractal Adaptive Moving Average
iGator	Gator Oscillator
iIchimoku	Ichimoku Kinko Hyo
iBWMFI	Market Facilitation Index by Bill Williams
iMomentum	Momentum
iMFI	Money Flow Index
iMA	Moving Average
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
iRVI	Relative Vigor Index
iStdDev	Standard Deviation
iStochastic	Stochastic Oscillator
iTEMA	Triple Exponential Moving Average
iTriX	Triple Exponential Moving Averages Oscillator
iWPR	Williams' Percent Range

iVIDyA	Variable Index DYnamic Average
iVolumes	Volumes

iAC

La función crea el indicador Accelerator Oscillator en la caché global del terminal de cliente y devuelve su manejador. Tiene sólo un búfer.

```
int  iAC(
   string           symbol,     // nombre del símbolo
   ENUM_TIMEFRAMES  period      // período
   );
```

Parámetros

symbol

[in] Símbolo de un instrumento financiero, cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso de fallo devuelve INVALID_HANDLE. Para liberar la memoria del ordenador de un indicador que ya no se usa, se usa la función IndicatorRelease() a la que se le pasa el manejador de este indicador.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                     Demo_iAC.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iAC."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- Construcción de iAC
#property indicator_label1  "iAC"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
```

```
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAC,               // usar iAC
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iAC;          // tipo de función
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
//--- búfers indicadores
double iACBuffer[];
double iACColors[];
//--- variable para guardar el manejador del indicador iAC
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Accelerator Oscillator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,iACBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iACColors,INDICATOR_COLOR_INDEX);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iAC)
      handle=iAC(name,period);
   else
      handle=IndicatorCreate(name,period,IND_AC);
//--- si no se puede crear el manejador
```

```
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iAC para el par %s/%s, c
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos para qué par símbolo/período ha sido calculado el indicador Acceler
   short_name=StringFormat("iAC(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iAC
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iACBuffer es más grande que los números de valores en el indi
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
```

```
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays iACBuffer y iACColors con los valores desde el indicador Ac
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArraysFromBuffer(iACBuffer,iACColors,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Accelerator Oscillator
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iAC           |
//+------------------------------------------------------------------+
bool FillArraysFromBuffer(double &values[],        // búfer indicador de valores Acce
                          double &color_indexes[], // búfer de color (para almacenar
                          int ind_handle,          // manejador del indicador iAC
                          int amount               // número de valores a copiar
                          )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iACBuffer con los valores desde el búfer indicador
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iAC, código del error
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- ahora copiaremos los índices de colores
   if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
     {
      //--- si el copiado ha fallado, mostramos el código del error
      PrintFormat("Fallo al copiar los valores de los colores desde el indicador iAC,
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
```

```
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iAD

Devuelve el manejador del indicador Accumulation/Distribution. Tiene sólo un búfer.

```
int  iAD(
   string               symbol,             // nombre del símbolo
   ENUM_TIMEFRAMES      period,             // período
   ENUM_APPLIED_VOLUME  applied_volume      // tipo de volumen para el cálculo
   );
```

Parámetros

symbol

[in] Símbolo de un instrumento financiero, cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

applied_volume

[in] Volumen usado. Puede ser cualquier valor de la enumeración ENUM_APPLIED_VOLUME.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                    Demo_iAD.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iAD."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iAD
#property indicator_label1  "iAD"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAD,               // usar iAD
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iAD;          // tipo de función
input ENUM_APPLIED_VOLUME  volumes;                // volumen que se utiliza
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
//--- búfer indicador
double         iADBuffer[];
//--- variable para guardar el manejador del indicador iAD
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Accumulation/Distribut
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iADBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iAD)
      handle=iAD(name,period,volumes);
   else
     {
```

```
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=volumes;
      handle=IndicatorCreate(name,period,IND_AD,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iAD para el par %s/%s, c
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos para qué par símbolo/período ha sido calculado el indicador Accumul
   short_name=StringFormat("iAD(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iAD
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
```

```
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iADBuffer supera el número de valores en el indicador iAD par
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iADBuffer con los valores desde el indicador Accumulation/Dis
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArrayFromBuffer(iADBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Accumulation/Distribution
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iAD           |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // búfer indicador de la línea Accumulat
                         int ind_handle,     // manejador del indicador iAD
                         int amount          // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- lenamos la parte del array iADBuffer con los valores desde el búfer indicador b
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iAD, código del error
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
```

```
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iADX

Devuelve el manejador del indicador Average Directional Movement Index.

```
int  iADX(
   string           symbol,          // nombre del símbolo
   ENUM_TIMEFRAMES  period,          // período
   int              adx_period       // período promedio
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

adx_period

[in] Período de cálculo del índice.

Nota

Números de búfers: 0 - MAIN_LINE, 1 - PLUSDI_LINE, 2 - MINUSDI_LINE.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iADX.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iADX."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
//--- plot ADX
```

```
#property indicator_label1  "ADX"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot DI_plus
#property indicator_label2  "DI_plus"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrYellowGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot DI_minus
#property indicator_label3  "DI_minus"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrWheat
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iADX,              // usar iADX
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iADX;         // tipo de función
input int                  adx_period=14;          // período de cálculo
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
//--- búfers indicadores
double         ADXBuffer[];
double         DI_plusBuffer[];
double         DI_minusBuffer[];
//--- variable para guardar el manejador del indicador iADX
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Average Directional Mo
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de arrays a los búfers indicadores
```

```
   SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iADX)
      handle=iADX(name,period,adx_period);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=adx_period;
      handle=IndicatorCreate(name,period,IND_ADX,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iADX para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos para qué par símbolo/período ha sido calculado el indicador Average
   short_name=StringFormat("iADX(%s/%s period=%d)",name,EnumToString(period),adx_peri
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iADX
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array ADXBuffer supera el número de valores en el indicador iADX pa
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array con los valores desde el indicador Average Directional Moveme
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArraysFromBuffers(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Average Directional Movement
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

```
//| Llenamos los búfers indicadores desde el indicador iADX          |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &adx_values[],      // búfer indicador de la línea
                           double &DIplus_values[],   // búfer indicador para DI+
                           double &DIminus_values[],  // búfer indicador para DI-
                           int ind_handle,            // manejador del indicador iADX
                           int amount                 // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array ADXBuffer con los valores desde el búfer indicador
   if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iADX, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array DI_plusBuffer con los valores desde el búfer indic
   if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iADX, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array DI_plusBuffer con los valores desde el búfer indic
   if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iADX, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iADXWilder

Devuelve el manejador del indicador Average Directional Movement Index by Welles Wilder.

```
int  iADXWilder(
   string           symbol,          // nombre del símbolo
   ENUM_TIMEFRAMES  period,          // período
   int              adx_period       // período promedio
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

adx_period

[in] Período de cálculo del índice.

Nota

Números de búfers: 0 - MAIN_LINE, 1 - PLUSDI_LINE, 2 - MINUSDI_LINE.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                  iADXWilder.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iADXWilder
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 3
#property indicator_plots   3
//--- plot ADX
```

```
#property indicator_label1  "ADX"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot DI_plus
#property indicator_label2  "DI_plus"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrYellowGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot DI_minus
#property indicator_label3  "DI_minus"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrWheat
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iADXWilder,         // usar iADXWilder
   Call_IndicatorCreate     // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iADXWilder;   // tipo de función
input int                  adx_period=14;          // período de cálculo
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
//--- búfers indicadores
double         ADXBuffer[];
double         DI_plusBuffer[];
double         DI_minusBuffer[];
//--- variable para guardar el manejador del indicador iADXWilder
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Average Directional Mo
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de arrays a los búfers indicadores
```

```
   SetIndexBuffer(0,ADXBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,DI_plusBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,DI_minusBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iADXWilder)
      handle=iADXWilder(name,period,adx_period);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=adx_period;
      handle=IndicatorCreate(name,period,IND_ADXW,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iADXWilder para el par %
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos para qué par símbolo/período se calcula el indicador Average Direct
   short_name=StringFormat("iADXWilder(%s/%s period=%d)",name,EnumToString(period),ad
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iADXWilder
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array ADXBuffer supera el número de valores en el indicador iADXWil
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array con los valores desde el indicador Average Directional Moveme
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArraysFromBuffers(ADXBuffer,DI_plusBuffer,DI_minusBuffer,handle,values_to_
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Average Directional Movement
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

```
//| Llenamos los búfers indicadores desde el indicador iADXWilder    |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &adx_values[],      // búfer indicador de la línea
                           double &DIplus_values[],   // búfer indicador para DI+
                           double &DIminus_values[],  // búfer indicador para DI-
                           int ind_handle,            // manejador del indicador iADX
                           int amount                 // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array ADXBuffer con los valores desde el búfer indicador
   if(CopyBuffer(ind_handle,0,0,amount,adx_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iADXWilder, código de
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array DI_plusBuffer con los valores desde el búfer indic
   if(CopyBuffer(ind_handle,1,0,amount,DIplus_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iADXWilder, código de
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array DI_plusBuffer con los valores desde el búfer indic
   if(CopyBuffer(ind_handle,2,0,amount,DIminus_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iADXWilder, código de
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iAlligator

Devuelve el manejador del indicador Alligator.

```
int  iAlligator(
   string               symbol,           // nombre del símbolo
   ENUM_TIMEFRAMES      period,           // período
   int                  jaw_period,       // período para el cálculo de mandíbulas
   int                  jaw_shift,        // desplazamiento horizontal de mandíbulas
   int                  teeth_period,     // período para el cálculo de dientes
   int                  teeth_shift,      // desplazamiento horizontal de dientes
   int                  lips_period,      // período para el cálculo de labios
   int                  lips_shift,       // desplazamiento horizontal de labios
   ENUM_MA_METHOD       ma_method,        // tipo de suavizado
   ENUM_APPLIED_PRICE   applied_price     // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

jaw_period

[in] Período promedio para la línea azul (mandíbulas de aligátor).

jaw_shift

[in] Desplazamiento de la línea azul con relación al gráfico de precios.

teeth_period

[in] Período promedio para la línea roja (dientes de aligátor).

teeth_shift

[in] Desplazamiento de la línea roja con relación al gráfico de precios.

lips_period

[in] Período promedio para la línea verde (labios de aligátor).

lips_shift

[in] Desplazamiento de la línea verde con relación al gráfico de precios.

ma_method

[in] Método de promedio. Puede ser cualquier valor de la enumeración ENUM_MA_METHOD.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Números de búfers: 0 - GATORJAW_LINE, 1 - GATORTEETH_LINE, 2 - GATORLIPS_LINE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_iAlligator.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iAlligator
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Alligator est

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   3
//--- plot Jaws
#property indicator_label1  "Jaws"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- plot Teeth
#property indicator_label2  "Teeth"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- plot Lips
#property indicator_label3  "Lips"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrLime
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
```

```
  {
   Call_iAlligator,         // usar iAlligator
   Call_IndicatorCreate     // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iAlligator;   // tipo de función
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
input int                  jaw_period=13;          // período para la línea de Mandíb
input int                  jaw_shift=8;            // desplazamiento de la línea de M
input int                  teeth_period=8;         // período para la línea de Diente
input int                  teeth_shift=5;          // desplazamiento de la línea de D
input int                  lips_period=5;          // período para la línea de Labios
input int                  lips_shift=3;           // desplazamiento de la línea de L
input ENUM_MA_METHOD       MA_method=MODE_SMMA;    // método de promediación de las l
input ENUM_APPLIED_PRICE   applied_price=PRICE_MEDIAN;// tipo del precio partiendo de
//--- búfers indicadores
double         JawsBuffer[];
double         TeethBuffer[];
double         LipsBuffer[];
//--- variable para guardar el manejador del indicador iAlligator
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Alligator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de arrays a los búfers indicadores
   SetIndexBuffer(0,JawsBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,TeethBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,LipsBuffer,INDICATOR_DATA);
//--- estableceremos el desplazamiento para cada línea
   PlotIndexSetInteger(0,PLOT_SHIFT,jaw_shift);
   PlotIndexSetInteger(1,PLOT_SHIFT,teeth_shift);
   PlotIndexSetInteger(2,PLOT_SHIFT,lips_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
```

```
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iAlligator)
      handle=iAlligator(name,period,jaw_period,jaw_shift,teeth_period,
                        teeth_shift,lips_period,lips_shift,MA_method,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[8];
     //--- períodos y desplazamientos de las líneas del Alligator
      pars[0].type=TYPE_INT;
      pars[0].integer_value=jaw_period;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=jaw_shift;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=teeth_period;
      pars[3].type=TYPE_INT;
      pars[3].integer_value=teeth_shift;
      pars[4].type=TYPE_INT;
      pars[4].integer_value=lips_period;
      pars[5].type=TYPE_INT;
      pars[5].integer_value=lips_shift;
//--- tipo de suavizado
      pars[6].type=TYPE_INT;
      pars[6].integer_value=MA_method;
//--- tipo del precio
      pars[7].type=TYPE_INT;
      pars[7].integer_value=applied_price;
//--- crearemos el manejador
      handle=IndicatorCreate(name,period,IND_ALLIGATOR,8,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iAlligator para el par %
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador Al
   short_name=StringFormat("iAlligator(%s/%s, %d,%d,%d,%d,%d,%d)",name,EnumToString(p
                           jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,
```

```
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iAlligator
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array JawsBuffer supera el número de valores en el indicador iAllig
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador Alligator
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArraysFromBuffers(JawsBuffer,jaw_shift,TeethBuffer,teeth_shift,LipsBuffer,
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
```

```
                              short_name,
                              values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Alligator
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iAlligator    |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &jaws_buffer[],  // búfer indicador para la línea d
                           int j_shift,            // desplazamiento de la línea de M
                           double &teeth_buffer[], // búfer indicador para la línea d
                           int t_shift,            // desplazamiento de la línea de D
                           double &lips_buffer[],  // búfer indicador para la línea d
                           int l_shift,            // desplazamiento de la línea de L
                           int ind_handle,         // manejador del indicador iAlliga
                           int amount              // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array JawsBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,-j_shift,amount,jaws_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iAlligator, código de
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array TeethBuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,1,-t_shift,amount,teeth_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iAlligator, código de
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array LipsBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,2,-l_shift,amount,lips_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iAlligator, código de
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
```

```
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iAMA

Devuelve el manejador del indicador Adaptive Moving Average. Tiene sólo un búfer.

```
int  iAMA(
   string               symbol,              // nombre del símbolo
   ENUM_TIMEFRAMES      period,              // período
   int                  ama_period,          // período AMA
   int                  fast_ma_period,      // período de la media móvil rápida
   int                  slow_ma_period,      // período de la media móvil lenta
   int                  ama_shift,           // desplazamiento horizontal del indicador
   ENUM_APPLIED_PRICE   applied_price        // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ama_period

[in] Período de cálculo durante el cual se calcula el coeficiente de efectividad.

fast_ma_period

[in] Período rápido para el cálculo del coeficiente de suavizado para un mercado rápido.

slow_ma_period

[in] Período lento para el cálculo del coeficiente de suavizado en ausencia de la tendencia.

ama_shift

[in] Desplazamiento del indicador con relación al precio del gráfico.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iAMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
```

```
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iAMA."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los de la AMA estánda

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iAMA
#property indicator_label1  "iAMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAMA,              // usar iAMA
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iAMA;          // tipo de función
input string                symbol=" ";              // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;   // período
input int                   ama_period=15;           // período para el cálculo
input int                   fast_ma_period=2;        // período de la MM rápida
input int                   slow_ma_period=30;       // período de la MM lenta
input int                   ama_shift=0;             // desplazamiento horizontal
input ENUM_APPLIED_PRICE    applied_price;           // tipo de precio
//--- búfer indicador
double         iAMABuffer[];
//--- variable para guardar el manejador del indicador iAMA
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Adaptive Moving Averag
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
```

```
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iAMABuffer,INDICATOR_DATA);
//--- fijaremos el desplazamiento
   PlotIndexSetInteger(0,PLOT_SHIFT,ama_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iAMA)
      handle=iAMA(name,period,ama_period,fast_ma_period,slow_ma_period,ama_shift,appl
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[5];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ama_period;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=fast_ma_period;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=slow_ma_period;
      pars[3].type=TYPE_INT;
      pars[3].integer_value=ama_shift;
      //--- tipo de precio
      pars[4].type=TYPE_INT;
      pars[4].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_AMA,5,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iAMA para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
```

```
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador Ad
   short_name=StringFormat("iAMA(%s/%s,%d,%d,%d,d)",name,EnumToString(period),ama_per
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iAlligator
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iADBuffer supera el número de valores en el indicador iAD par
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador Alligator
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArrayFromBuffer(iAMABuffer,ama_shift,handle,values_to_copy)) return(0);
//--- creamos el mensaje
```

```
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Accelerator Oscillator
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iAMA              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &ama_buffer[],  // búfer indicador de la línea AMA
                         int a_shift,           // desplazamiento de la línea AMA
                         int ind_handle,        // manejador del indicador iAMA
                         int amount             // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iAMABuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,-a_shift,amount,ama_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iAlligator, código de
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iAO

Devuelve el manejador del indicador Awesome Oscillator. Tiene sólo un búfer.

```
int  iAO(
   string           symbol,     // nombre del símbolo
   ENUM_TIMEFRAMES  period      // período
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                     Demo_iAO.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iAO."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- Construcción de iAO
#property indicator_label1  "iAO"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen,clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
```

```
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iAO,               // usar iAO
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iAO;          // tipo de función
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
//--- búfers indicadores
double         iAOBuffer[];
double         iAOColors[];
//--- variable para guardar el manejador del indicador iAO
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Awesome Oscillator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,iAOBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iAOColors,INDICATOR_COLOR_INDEX);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iAO)
      handle=iAO(name,period);
   else
      handle=IndicatorCreate(name,period,IND_AO);
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
```

```
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iAO para el par %s/%s, c
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos para qué par símbolo/período ha sido calculado el indicador Awesome
   short_name=StringFormat("iAO(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iAO
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iAOBuffer supera el número de valores en el indicador iAO par
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
```

```
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays iAOBuffer y iAOColors con los valores desde el indicador Aw
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArraysFromBuffer(iAOBuffer,iAOColors,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Accelerator Oscillator
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iAO           |
//+------------------------------------------------------------------+
bool FillArraysFromBuffer(double &values[],        // búfer indicador de los valores
                          double &color_indexes[], // búfer de color (para almacenar
                          int ind_handle,          // manejador del indicador iAO
                          int amount               // número de valores a copiar
                          )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iAOBuffer con los valores desde el búfer indicador
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iAO, código del error
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- ahora copiaremos los índices de colores
   if(CopyBuffer(ind_handle,1,0,amount,color_indexes)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los valores de los colores desde el indicador iAO,
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
```

```
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iATR

Devuelve el manejador del indicador Average True Range. Tiene sólo un búfer.

```
int  iATR(
   string           symbol,        // nombre del símbolo
   ENUM_TIMEFRAMES  period,        // período
   int              ma_period      // período promedio
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para el cálculo del indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iATR.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iATR."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- plot iATR
#property indicator_label1  "iATR"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iATR,// usar iATR
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input int                  atr_period=14;          // período para el cálculo
input Creation             type=Call_iATR;          // tipo de función
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
//--- búfer indicador
double         iATRBuffer[];
//--- variable para guardar el manejador del indicador iAC
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Average True Range
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iATRBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iATR)
      handle=iATR(name,period,atr_period);
   else
     {
```

```
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      pars[0].type=TYPE_INT;
      pars[0].integer_value=atr_period;
      handle=IndicatorCreate(name,period,IND_ATR,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iATR para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador Av
   short_name=StringFormat("iATR(%s/%s, period=%d)",name,EnumToString(period),atr_per
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iATR
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
```

```
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iATRBuffer supera el número de valores en el indicador iATR p
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iATRBuffer con los valores desde el indicador Average True Ra
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iATRBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Accelerator Oscillator
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iATR              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // búfer indicador de valores ATR
                         int ind_handle,    // manejador del indicador iATR
                         int amount         // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iATRBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iATR, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
```

```
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iBearsPower

Devuelve el manejador del indicador Bears Power. Tiene sólo un búfer.

```
int  iBearsPower(
   string              symbol,           // nombre del símbolo
   ENUM_TIMEFRAMES     period,           // período
   int                 ma_period,        // período promedio
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para el cálculo del indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                              Demo_iBearsPower.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iBearsPowe
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- Construcción de iBearsPower
#property indicator_label1  "iBearsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBearsPower,       // usar iBearsPower
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iBearsPower;  // tipo de función
input int                  ma_period=13;           // período de la media móvil
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
//--- búfer indicador
double         iBearsPowerBuffer[];
//--- variable para guardar el manejador del indicador iBearsPower
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Bears Power
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iBearsPowerBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iBearsPower)
      handle=iBearsPower(name,period,ma_period);
   else
     {
```

```
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_BEARS,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iBearsPower para el par
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador Be
   short_name=StringFormat("iBearsPower(%s/%s, period=%d)",name,EnumToString(period),
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iBearsPower
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
```

```
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iATRBuffer supera el número de valores en el indicador iBears
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iBearsPowerBuffer con los valores desde el indicador Bears Po
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iBearsPowerBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Bears Power
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iBearsPower       |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // el búfer indicador de valores Bears Po
                         int ind_handle,    // manejador del indicador iBearsPower
                         int amount         // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iBearsPowerBuffer con los valores desde el búfer i
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iBearsPower, código d
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iBands

Devuelve el manejador del indicador Bollinger Bands®.

```
int  iBands(
   string               symbol,           // nombre del símbolo
   ENUM_TIMEFRAMES      period,           // período
   int                  bands_period,     // período para el cálculo de la línea medi
   int                  bands_shift,      // desplazamiento horizontal del indicador
   double               deviation,        // número de desviaciones estándares
   ENUM_APPLIED_PRICE   applied_price     // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

bands_period

[in] Período promedio para la línea principal del indicador.

bands_shift

[in] Desplazamiento del indicador con relación al gráfico de precios.

deviation

[in] Desviación de la línea principal.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Números de búfers: 0 - BASE_LINE, 1 - UPPER_BAND, 2 - LOWER_BAND

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iBands.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
```

```
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iBands."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_chart_window
#property indicator_buffers 3
#property indicator_plots   3
//--- construcción de Upper
#property indicator_label1  "Upper"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrMediumSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- construcción de Lower
#property indicator_label2  "Lower"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrMediumSeaGreen
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- construcción de Middle
#property indicator_label3  "Middle"
#property indicator_type3   DRAW_LINE
#property indicator_color3  clrMediumSeaGreen
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBands,             // usar iBands
   Call_IndicatorCreate     // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iBands;          // tipo de función
input int                  bands_period=20;           // período de la media móvil
input int                  bands_shift=0;             // sangría
input double               deviation=2.0;             // número de desviaciones están
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // tipo de precio
input string               symbol=" ";                // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // período de tiempo
//--- búfers indicadores
double         UpperBuffer[];
double         LowerBuffer[];
```

```
double         MiddleBuffer[];
//--- variable para guardar el manejador del indicador iBands
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Bollinger Bands
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
   SetIndexBuffer(2,MiddleBuffer,INDICATOR_DATA);
//--- estableceremos el desplazamiento para cada línea
   PlotIndexSetInteger(0,PLOT_SHIFT,bands_shift);
   PlotIndexSetInteger(1,PLOT_SHIFT,bands_shift);
   PlotIndexSetInteger(2,PLOT_SHIFT,bands_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iBands)
      handle=iBands(name,period,bands_period,bands_shift,deviation,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[4];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=bands_period;
      //--- desplazamiento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=bands_shift;
      //--- número de desviaciones estándares
      pars[2].type=TYPE_DOUBLE;
```

```
      pars[2].double_value=deviation;
      //--- tipo de precio
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_BANDS,4,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iBands para el par %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos para qué par símbolo/período ha sido calculado el indicador Bolling
   short_name=StringFormat("iBands(%s/%s, %d,%d,%G,%s)",name,EnumToString(period),
                           bands_period,bands_shift,deviation,EnumToString(applied_pr
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iBands
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
```

```
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el tamaño de los arrays indicadores supera el número de valores en el
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array con los valores desde el indicador Bollinger Bands
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArraysFromBuffers(MiddleBuffer,UpperBuffer,LowerBuffer,bands_shift,handle,
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Bollinger Bands
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iBands        |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &base_values[],     // búfer indicador de la línea
                           double &upper_values[],    // búfer indicador del borde su
                           double &lower_values[],    // búfer indicador del borde in
                           int shift,                 // desplazamiento
                           int ind_handle,            // manejador del indicador iBan
                           int amount                 // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array MiddleBuffer con los valores desde el búfer indica
   if(CopyBuffer(ind_handle,0,-shift,amount,base_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iBands, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
```

```
      return(false);
     }

//--- llenamos una parte del array UpperBuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,1,-shift,amount,upper_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iBands, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array LowerBuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,2,-shift,amount,lower_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iBands, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iBullsPower

Devuelve el manejador del indicador Bulls Power. Tiene sólo un búfer.

```
int  iBullsPower(
   string              symbol,           // nombre del símbolo
   ENUM_TIMEFRAMES     period,           // período
   int                 ma_period,        // período promedio
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para el cálculo del indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                              Demo_iBullsPower.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iBullsPowe
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iBullsPower
#property indicator_label1  "iBullsPower"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBullsPower,       // usar iBullsPower
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iBullsPower;  // tipo de función
input int                  ma_period=13;           // período de la media móvil
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
//--- búfer indicador
double         iBullsPowerBuffer[];
//--- variable para guardar el manejador del indicador iBullsPower
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Bulls Power
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iBullsPowerBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iBullsPower)
      handle=iBullsPower(name,period,ma_period);
   else
     {
```

```
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_BULLS,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iBullsPower para el par
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador Bu
   short_name=StringFormat("iBullsPower(%s/%s, period=%d)",name,EnumToString(period),
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iBullsPower
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
```

```
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iBullsPowerBuffer supera el número de valores en el indicador
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iBullsPowerBuffer con los valores desde el indicador Bulls Po
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iBullsPowerBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Bulls Power
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iBullsPower       |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // el búfer indicador de valores Bulls Po
                         int ind_handle,    // manejador del indicador iBullsPower
                         int amount         // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iBullsPowerBuffer con los valores desde el búfer i
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iBullsPower, código d
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
//+------------------------------------------------------------------+
```

iCCI

Devuelve el manejador del indicador Commodity Channel Index. Tiene sólo un búfer.

```
int  iCCI(
   string               symbol,            // nombre del símbolo
   ENUM_TIMEFRAMES      period,            // período
   int                  ma_period,         // período promedio
   ENUM_APPLIED_PRICE   applied_price      // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para el cálculo del indicador.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iCCI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iCCI."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
```

```
//--- construcción de iCCI
#property indicator_label1  "iCCI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- niveles horizontales en la ventana del indicador
#property indicator_level1  -100.0
#property indicator_level2  100.0
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iCCI,              // usar iCCI
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iCCI;               // tipo de función
input int                  ma_period=14;                 // período de la media móvil
input ENUM_APPLIED_PRICE   applied_price=PRICE_TYPICAL;  // tipo de precio
input string               symbol=" ";                   // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- búfer indicador
double         iCCIBuffer[];
//--- variable para guardar el manejador del indicador iCCI
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Commodity Channel Inde
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iCCIBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
```

```
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iCCI)
      handle=iCCI(name,period,ma_period,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[2];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- tipo de precio
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_CCI,2,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iCCI para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos para qué par símbolo/período ha sido calculado el indicador Bolling
   short_name=StringFormat("iCCI(%s/%s, %d, %s)",name,EnumToString(period),
                           ma_period,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iCCI
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iATRBuffer supera el número de valores en el indicador iCCI p
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iCCIBuffer con los valores desde el indicador Commodity Chann
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iCCIBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Commodity Channel Index
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iCCI              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // búfer indicador de valores Commodity C
                         int ind_handle,    // manejador del indicador iCCI
                         int amount         // número de valores a copiar
                         )
```

```
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iCCIBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iCCI, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iChaikin

Devuelve el manejador del indicador Chaikin Oscillator. Tiene sólo un búfer.

```
int  iChaikin(
   string               symbol,              // nombre del símbolo
   ENUM_TIMEFRAMES      period,              // período
   int                  fast_ma_period,      // período rápido
   int                  slow_ma_period,      // período lento
   ENUM_MA_METHOD       ma_method,           // tipo de suavizado
   ENUM_APPLIED_VOLUME  applied_volume       // valor utilizado
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

fast_ma_period

[in] Período promedio rápido para el cálculo del indicador.

slow_ma_period

[in] Período promedio lento para el cálculo del indicador.

ma_method

[in] Tipo de promedio. Puede ser una de las constantes del promedio ENUM_MA_METHOD.

applied_volume

[in] Valor utilizado. Puede ser una de las constantes de ENUM_APPLIED_VOLUME.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_iChaikin.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iChaikin."
```

```
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iChaikin
#property indicator_label1  "iChaikin"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iChaikin,          // usar iChaikin
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iChaikin;          // tipo de función
input int                  fast_ma_period=3;            // período rápido
input int                  slow_ma_period=10;           // período lento
input ENUM_MA_METHOD       ma_method=MODE_EMA;          // tipo de suavizado
input ENUM_APPLIED_VOLUME  applied_volume=VOLUME_TICK;  // tipo de volumen
input string               symbol=" ";                  // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;       // timeframe
//--- búfer indicador
double         iChaikinBuffer[];
//--- variable para guardar el manejador del indicador iChaikin
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Chaikin Oscillator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iChaikinBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
```

```
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iChaikin)
      handle=iChaikin(name,period,fast_ma_period,slow_ma_period,ma_method,applied_vol
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[4];
      //--- período rápido
      pars[0].type=TYPE_INT;
      pars[0].integer_value=fast_ma_period;
      //--- período lento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=slow_ma_period;
      //--- tipo de suavizado
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- tipo de volumen
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_CHAIKIN,4,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iChaikin para el par %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos para qué par símbolo/período ha sido calculado el indicador Chaikin
   short_name=StringFormat("iChaikin(%s/%s, %d, %d, %s, %s)",name,EnumToString(period
                           fast_ma_period,slow_ma_period,
                           EnumToString(ma_method),EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iChaikin
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iChaikinBuffer supera el número de valores en el indicador iC
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iChaikinBuffer con los valores desde el indicador Chaikin Osc
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iChaikinBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
```

```
   Comment(comm);
//--- recordaremos el número de valores en el indicador Chaikin Oscillator
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iChaikin          |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // búfer indicador de valores Chaikin Osc
                         int ind_handle,    // manejador del indicador iChaikin
                         int amount         // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iChaikinBuffer con los valores desde el búfer indi
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iChaikin, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iCustom

Devuelve el manejador del indicador personalizado especificado.

```
int  iCustom(
   string           symbol,     // nombre del símbolo
   ENUM_TIMEFRAMES  period,     // período
   string           name        // carpeta/nombre_de_indicador personalizado
   ...                          // lista de parámetros de entrada del indicador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

name

[in] Nombre del indicador personalizado que contiene la ruta respecto al directorio raíz de indicadores (MQL5/Indicators/). Si un indicador se encuentra en un subdirectorio, por ejemplo, en MQL5/Indicators/**Examples**, entonces su nombre ha de ser como sigue - "**Examples**\\nombre_de_indicador" (es obligatorio el uso de dos barras inversas en vez de una como un separador).

...

[in] input-parámetros del indicador personalizado están separados por comas. El tipo y el orden de seguimiento de parámetros deben corresponder. Si los parámetros no están especificados, entonces se usarán los valores por defecto.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

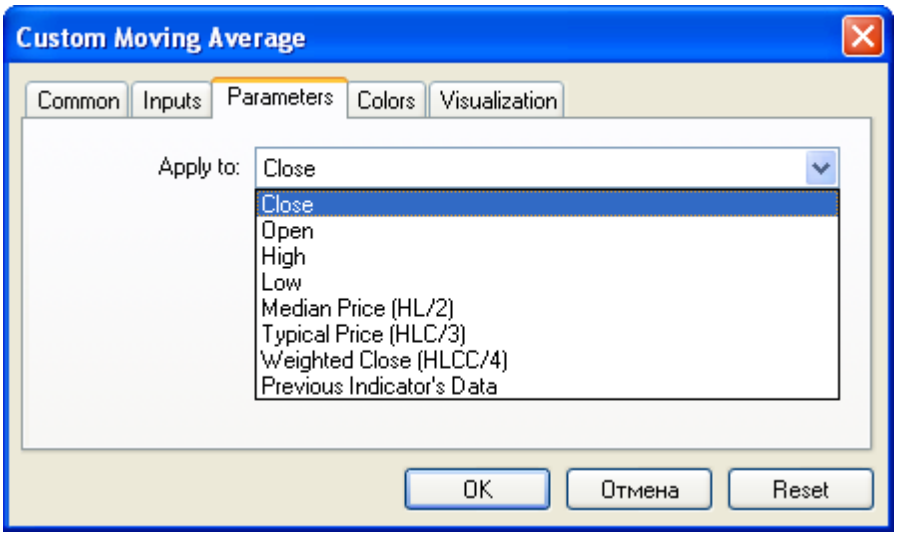
Un indicador personalizado tiene que estar compilado (un archivo con la extensión EX5) y debe estar ubicado en el directorio MQL5/Indicators del terminal de cliente o en una de sus subcarpetas.

Los indicadores que requieren verificaciones se definen automáticamente desde la llamada a la función iCustom(), si el parámetro correspondiente es fijado por una cadena constante. Para los demás casos (uso de función IndicatorCreate() o uso de una cadena no constante en el parámetro que asigna el nombre del indicador) esta propiedad #property tester_indicator es necesaria:

```
#property tester_indicator "indicator_name.ex5"
```

Si en el indicador se usa la forma de llamada, entonces al iniciar el indicador personalizado en la pestaña "Parameters" podemos especificar adicionalmente los datos a base de los cuales va a ser

calculado. Si el parámetro "Apply to" no está elegido explícitamente, por defecto el cálculo se realiza a base de los valores "Close".



Cuando un indicador personalizado se llama desde el programa mql5, el parámetro Applied_Price o un manejador de otro indicador debe ser pasado el último después de todos los parámetros de entrada previstos por el indicador personalizado.

Véase también

Propiedades de programas, Acceso a las series temporales y a los datos de indicadores, IndicatorCreate(), IndicatorRelease()

Ejemplo:

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   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
//--- input parameters
input int MA_Period=21;
input int MA_Shift=0;
input ENUM_MA_METHOD MA_Method=MODE_SMA;
//--- indicator buffers
double         Label1Buffer[];
//--- manejador del indicador personalizado Custom Moving Average.mq5
int MA_handle;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
```

```
//--- indicator buffers mapping
   SetIndexBuffer(0,Label1Buffer,INDICATOR_DATA);
   ResetLastError();
   MA_handle=iCustom(NULL,0,"Examples\\Custom Moving Average",
                     MA_Period,
                     MA_Shift,
                     MA_Method,
                     PRICE_CLOSE // calculamos según los precios de cierre
                     );
   Print("MA_handle = ",MA_handle,"  error = ",GetLastError());
//---
   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[])
  {
//--- vamos a copiar los valores del indicador Custom Moving Average a nuestro búfer
   int copy=CopyBuffer(MA_handle,0,0,rates_total,Label1Buffer);
   Print("copy = ",copy,"    rates_total = ",rates_total);
//--- si el intento es fallido, avisemos de ello
   if(copy<=0)
      Print("Fallo al obtener los valores del indicador Custom Moving Average");
//--- return value of prev_calculated for next call
   return(rates_total);
  }
//+------------------------------------------------------------------+
```

iDEMA

Devuelve el manejador del indicador Double Exponential Moving Average. Tiene sólo un búfer.

```
int  iDEMA(
   string               symbol,            // nombre del símbolo
   ENUM_TIMEFRAMES      period,            // período
   int                  ma_period,         // período promedio
   int                  ma_shift,          // desplazamiento horizontal del indicador
   ENUM_APPLIED_PRICE   applied_price      // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio (cantidad de barras) para el cálculo del indicador.

ma_shift

[in] Desplazamiento del indicador con relación al gráfico de precios.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iDEMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iDEMA."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
```

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iDEMA
#property indicator_label1  "iDEMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iDEMA,              // usar iDEMA
   Call_IndicatorCreate     // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iDEMA;           // tipo de función
input int                   ma_period=14;              // período de la media móvil
input int                   ma_shift=0;                // desplazamiento
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // tipo de precio
input string                symbol=" ";                // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double         iDEMABuffer[];
//--- variable para guardar el manejador del indicador iDEMA
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Double Exponential Mov
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iDEMABuffer,INDICATOR_DATA);
   //--- fijaremos el desplazamiento
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
```

```
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iDEMA)
      handle=iDEMA(name,period,ma_period,ma_shift,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[3];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- desplazamiento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- tipo de precio
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_DEMA,3,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iDEMA para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador Do
   short_name=StringFormat("iDEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iDEMA
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iDEMABuffer supera el número de valores en el indicador iDEMA
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iDEMABuffer con los valores desde el indicador Double Exponen
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iDEMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Double Exponential Moving Ave
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
```

```
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iDEMA             |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // búfer indicador de valores Double Expo
                         int shift,         // desplazamiento
                         int ind_handle,    // manejador del indicador iDEMA
                         int amount         // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iDEMABuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iDEMA, código del err
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iDeMarker

Devuelve el manejador del indicador DeMarker. Tiene sólo un búfer.

```
int  iDeMarker(
   string           symbol,        // nombre del símbolo
   ENUM_TIMEFRAMES  period,        // período
   int              ma_period      // período promedio
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para el cálculo del indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_iDeMarker.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iDeMarker."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iDeMarker
#property indicator_label1  "iDeMarker"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- niveles horizontales en la ventana del indicador
#property indicator_level1  0.3
#property indicator_level2  0.7
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iDeMarker,         // usar iDeMarker
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iDeMarker;       // tipo de función
input int                  ma_period=14;              // período de la media móvil
input string               symbol=" ";                // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double         iDeMarkerBuffer[];
//--- variable para guardar el manejador del indicador iDeMarker
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador DeMarker
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iDeMarkerBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iDeMarker)
```

```
      handle=iDeMarker(name,period,ma_period);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_DEMARKER,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iDeMarker para el par %s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador De
   short_name=StringFormat("iDeMarker(%s/%s, period=%d)",name,EnumToString(period),ma
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iDeMarker
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
```

```
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iDEMABuffer supera el número de valores en el indicador iDeMa
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iDeMarkerBuffer con los valores desde el indicador DeMarker
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iDeMarkerBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador DeMarker
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iDeMarker         |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // búfer indicador de valores DeMarker
                         int ind_handle,    // manejador del indicador iDeMarker
                         int amount         // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iDeMarkerBuffer con los valores desde el búfer ind
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iDeMarker, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
```

```
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iEnvelopes

Devuelve el manejador del indicador Envelopes.

```
int  iEnvelopes(
   string               symbol,           // nombre del símbolo
   ENUM_TIMEFRAMES      period,           // período
   int                  ma_period,        // período para calcular la línea media
   int                  ma_shift,         // desplazamiento horizontal del indicador
   ENUM_MA_METHOD       ma_method,        // tipo de suavizado
   ENUM_APPLIED_PRICE   applied_price,    // tipo de precio o manejador
   double               deviation         // desviación de márgenes respecto a la lín
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio de la línea principal del indicador.

ma_shift

[in] Desplazamiento del indicador con relación al gráfico de precios.

ma_method

[in] Método de promedio. Puede ser cualquier valor de la enumeración ENUM_MA_METHOD.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

deviation

[in] Desviación de la línea principal en términos de porcientos.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Números de búfers: 0 - UPPER_LINE, 1 - LOWER_LINE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_iEnvelopes.mq5 |
```

```
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iEnvelopes
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//--- construcción de Upper
#property indicator_label1  "Upper"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- construcción de Lower
#property indicator_label2  "Lower"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iEnvelopes,        // usar iEnvelopes
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iEnvelopes;      // tipo de función
input int                  ma_period=14;              // período de la media móvil
input int                  ma_shift=0;                // desplazamiento
input ENUM_MA_METHOD       ma_method=MODE_SMA;        // tipo de suavizado
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // tipo de precio
input double               deviation=0.1;             // desviación de los márgenes d
input string               symbol=" ";                // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double         UpperBuffer[];
double         LowerBuffer[];
//--- variable para guardar el manejador del indicador iEnvelopes
```

```
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Envelopes
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,UpperBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,LowerBuffer,INDICATOR_DATA);
//--- estableceremos el desplazamiento para cada línea
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
   PlotIndexSetInteger(1,PLOT_SHIFT,ma_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iEnvelopes)
      handle=iEnvelopes(name,period,ma_period,ma_shift,ma_method,applied_price,deviat
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[5];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- desplazamiento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- tipo de suavizado
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- tipo de precio
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
```

```
      //--- tipo de precio
      pars[4].type=TYPE_DOUBLE;
      pars[4].double_value=deviation;
      handle=IndicatorCreate(name,period,IND_ENVELOPES,5,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iEnvelopes para el par %
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador En
   short_name=StringFormat("iEnvelopes(%s/%s, %d, %d, %s,%s, %G)",name,EnumToString(p
   ma_period,ma_shift,EnumToString(ma_method),EnumToString(applied_price),deviation);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iEnvelopes
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
```

```
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array UpperBuffer supera el número de valores en el indicador iEnve
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays UpperBuffer y LowerBuffer con los valores desde el indicado
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArraysFromBuffers(UpperBuffer,LowerBuffer,ma_shift,handle,values_to_copy))
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Envelopes
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iEnvelopes    |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &upper_values[],     // búfer indicador del borde su
                           double &lower_values[],     // búfer indicador del borde in
                           int shift,                  // desplazamiento
                           int ind_handle,             // manejador del indicador iEnv
                           int amount                  // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array UpperBuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,0,-shift,amount,upper_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iEnvelopes, código de
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
```

```
//--- llenamos una parte del array LowerBuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,1,-shift,amount,lower_values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iEnvelopes, código de
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iForce

Devuelve el manejador del indicador Force Index. Tiene sólo un búfer.

```
int  iForce(
   string               symbol,            // nombre del símbolo
   ENUM_TIMEFRAMES      period,            // período
   int                  ma_period,         // período promedio
   ENUM_MA_METHOD       ma_method,         // tipo de suavizado
   ENUM_APPLIED_VOLUME  applied_volume     // tipo de volumen para el cálculo
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para el cálculo del indicador.

ma_method

[in] Método de promedio. Puede ser cualquier valor de la enumeración ENUM_MA_METHOD.

applied_volume

[in] Volumen usado. Puede ser cualquier valor de la enumeración ENUM_APPLIED_VOLUME.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iForce.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iForce."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
```

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iForce
#property indicator_label1  "iForce"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iForce,            // usar iForce
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iForce;             // tipo de función
input int                  ma_period=13;                 // período promedio
input ENUM_MA_METHOD       ma_method=MODE_SMA;           // tipo de suavizado
input ENUM_APPLIED_VOLUME  applied_volume=VOLUME_TICK;   // tipo de volumen
input string               symbol=" ";                   // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- búfer indicador
double         iForceBuffer[];
//--- variable para guardar el manejador del indicador iForce
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Force
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iForceBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
```

```
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iForce)
      handle=iForce(name,period,ma_period,ma_method,applied_volume);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[3];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- tipo de suavizado
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_method;
      //--- tipo de volumen
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_volume;
      //--- tipo de precio
      handle=IndicatorCreate(name,period,IND_FORCE,3,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iForce para el par %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador Fo
   short_name=StringFormat("iForce(%s/%s, %d, %s, %s)",name,EnumToString(period),
                           ma_period,EnumToString(ma_method),EnumToString(applied_vol
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iForce
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iForceBuffer supera el número de valores en el indicador iFor
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iForceBuffer con los valores desde el indicador Force
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iForceBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Force
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iForce            |
```

```
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // búfer indicador de valores Force Index
                         int ind_handle,    // manejador del indicador iForce
                         int amount         // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iForceBuffer con los valores desde el búfer indica
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iForce, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iFractals

Devuelve el manejador del indicador Fractals.

```
int  iFractals(
   string           symbol,     // nombre del símbolo
   ENUM_TIMEFRAMES  period      // período
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Números de búfers: 0 - UPPER_LINE, 1 - LOWER_LINE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_iFractals.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iFractals."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
#property indicator_chart_window
#property indicator_buffers 2
#property indicator_plots   2
//--- construcción de FractalUp
#property indicator_label1  "FractalUp"
```

```
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrBlue
//--- construcción de FractalDown
#property indicator_label2  "FractalDown"
#property indicator_type2   DRAW_ARROW
#property indicator_color2  clrRed
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iFractals,         // usar iFractals
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iFractals;          // tipo de función
input string               symbol=" ";                   // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- búfers indicadores
double         FractalUpBuffer[];
double         FractalDownBuffer[];
//--- variable para guardar el manejador del indicador iFractals
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Fractals
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,FractalUpBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,FractalDownBuffer,INDICATOR_DATA);
//--- estableceremos los códigos desde el conjunto Wingdings para las propiedades PLO
   PlotIndexSetInteger(0,PLOT_ARROW,217); // flecha arriba
   PlotIndexSetInteger(1,PLOT_ARROW,218); // flecha abajo
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
```

```
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iFractals)
      handle=iFractals(name,period);
   else
      handle=IndicatorCreate(name,period,IND_FRACTALS);
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iFractals para el par %s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador Fr
   short_name=StringFormat("iFractals(%s/%s)",name,EnumToString(period));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iFractals
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
```

```
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array FractalUpBuffer supera el número de valores en el indicador i
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays FractalUpBuffer y FractalDownBuffer con los valores desde e
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArraysFromBuffers(FractalUpBuffer,FractalDownBuffer,handle,values_to_copy)
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Fractals
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iFractals     |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &up_arrows[],        // búfer indicador de las flec
                           double &down_arrows[],      // búfer indicador de las flec
                           int ind_handle,             // manejador del indicador iFr
                           int amount                  // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array FractalUpBuffer con los valores desde el búfer ind
   if(CopyBuffer(ind_handle,0,0,amount,up_arrows)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iFractals al array Fr
                  GetLastError());
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
```

```
      return(false);
     }
//--- llenamos una parte del array FractalDownBuffer con los valores desde el búfer i
   if(CopyBuffer(ind_handle,1,0,amount,down_arrows)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iFractals al array Fr
                  GetLastError());
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iFrAMA

Devuelve el manejador del indicador Fractal Adaptive Moving Average. Tiene sólo un búfer.

```
int  iFrAMA(
   string               symbol,            // nombre del símbolo
   ENUM_TIMEFRAMES      period,            // período
   int                  ma_period,         // período promedio
   int                  ma_shift,          // desplazamiento horizontal del indicador
   ENUM_APPLIED_PRICE   applied_price      // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio (cantidad de barras) para el cálculo del indicador.

ma_shift

[in] Desplazamiento del indicador con relación al gráfico de precios.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                 Demo_iFrAMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iFrAMA."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
```

```
#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iFrAMA
#property indicator_label1  "iFrAMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iFrAMA,            // usar iFrAMA
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iFrAMA;             // tipo de función
input int                  ma_period=14;                 // período promedio
input int                  ma_shift=0;                   // desplazamiento
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE;    // tipo de precio
input string               symbol=" ";                   // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;        // timeframe
//--- búfer indicador
double         iFrAMABuffer[];
//--- variable para guardar el manejador del indicador iFrAMA
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Fractal Adaptive Movin
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iFrAMABuffer,INDICATOR_DATA);
//--- fijaremos el desplazamiento
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
```

```
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iFrAMA)
      handle=iFrAMA(name,period,ma_period,ma_shift,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[3];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- desplazamiento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- tipo de precio
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_price;
      //--- tipo de precio
      handle=IndicatorCreate(name,period,IND_FRAMA,3,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iFrAMA para el par %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos a base de qué par símbolo/período ha sido calculado el indicador iF
   short_name=StringFormat("iFrAMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iFrAMA
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iFrAMABuffer supera el número de valores en el indicador iFrA
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iFrAMABuffer con los valores desde el indicador Fractal Adapt
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iFrAMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Fractal Adaptive Moving Avera
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
```

```
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iFrAMA            |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // búfer indicador de valores Fractal Ada
                         int shift,         // desplazamiento
                         int ind_handle,    // manejador del indicador iFrAMA
                         int amount         // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iFrAMABuffer con los valores desde el búfer indica
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iFrAMA, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iGator

Devuelve el manejador del indicador Gator. El oscilador muestra la diferencia entre la línea azul y roja del Aligátor (histograma de arriba) y la diferencia entre la línea roja y verde (histograma de abajo).

```
int  iGator(
   string              symbol,           // nombre del símbolo
   ENUM_TIMEFRAMES     period,           // período
   int                 jaw_period,       // período para el cálculo de mandíbulas
   int                 jaw_shift,        // desplazamiento horizontal de mandíbulas
   int                 teeth_period,     // período para el cálculo de dientes
   int                 teeth_shift,      // desplazamiento horizontal de dientes
   int                 lips_period,      // período para el cálculo de labios
   int                 lips_shift,       // desplazamiento horizontal de labios
   ENUM_MA_METHOD      ma_method,        // tipo de suavizado
   ENUM_APPLIED_PRICE  applied_price     // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

jaw_period

[in] Período promedio para la línea azul (mandíbulas de aligátor).

jaw_shift

[in] Desplazamiento de la línea azul del Aligátor con relación al gráfico de precios. No tiene relación directa con el desplazamiento visual del histograma del indicador.

teeth_period

[in] Período promedio para la línea roja (dientes de aligátor).

teeth_shift

[in] Desplazamiento de la línea roja del Aligátor con relación al gráfico de precios. No tiene relación directa con el desplazamiento visual del histograma del indicador.

lips_period

[in] Período promedio para la línea verde (labios de aligátor).

lips_shift

[in] Desplazamiento de la línea verde del Aligátor con relación al gráfico de precios. No tiene relación directa con el desplazamiento visual del histograma del indicador.

ma_method

[in] Método de promedio. Puede ser cualquier valor de la enumeración ENUM_MA_METHOD.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Números de buffers: 0 - UPPER_HISTOGRAM, 1- buffer de color del histograma de arriba, 2 - LOWER_HISTOGRAM, 3- buffer de color del histograma de abajo.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iGator.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iGator."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Gator Oscilla

#property indicator_separate_window
#property indicator_buffers 4
#property indicator_plots   2
//--- construcción de GatorUp
#property indicator_label1  "GatorUp"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- construcción de GatorDown
#property indicator_label2  "GatorDown"
#property indicator_type2   DRAW_COLOR_HISTOGRAM
#property indicator_color2  clrGreen, clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
```

```
   Call_iGator,            // usar iGator
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iGator;       // tipo de función
input string               symbol=" ";             // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;  // período de tiempo
input int                  jaw_period=13;          // período para la línea de Mandíb
input int                  jaw_shift=8;            // desplazamiento de la línea de M
input int                  teeth_period=8;         // período para la línea de Diente
input int                  teeth_shift=5;          // desplazamiento de la línea de D
input int                  lips_period=5;          // período para la línea de Labios
input int                  lips_shift=3;           // desplazamiento de la línea de L
input ENUM_MA_METHOD       MA_method=MODE_SMMA;    // método de promediación de las l
input ENUM_APPLIED_PRICE   applied_price=PRICE_MEDIAN;// tipo del precio partiendo de
//--- búfers indicadores
double         GatorUpBuffer[];
double         GatorUpColors[];
double         GatorDownBuffer[];
double         GatorDownColors[];
//--- variable para guardar el manejador del indicador iGator
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- valores de desplazamiento para el histograma superior e inferior
int shift;
//--- vamos a guardar el número de los valores en el indicador Gator Oscillator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,GatorUpBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,GatorUpColors,INDICATOR_COLOR_INDEX);
   SetIndexBuffer(2,GatorDownBuffer,INDICATOR_DATA);
   SetIndexBuffer(3,GatorDownColors,INDICATOR_COLOR_INDEX);
/*
  ¡Todos los desplazamientos especificados en los parámetros se refieren al indicador
  ¡Por esta razón dichos desplazamientos no mueven el mismo indicador Gator, sino mue
  a base de cuyos valores se construyen los valores del indicador Gator Oscillator!
*/
//--- calcularemos el desplazamiento para el histograma superior e inferior que supon
   shift=MathMin(jaw_shift,teeth_shift);
   PlotIndexSetInteger(0,PLOT_SHIFT,shift);
```

```
//--- a pesar de que el indicador contiene dos histogramas, se utiliza el mismo despl
   PlotIndexSetInteger(1,PLOT_SHIFT,shift);

//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iGator)
      handle=iGator(name,period,jaw_period,jaw_shift,teeth_period,teeth_shift,
                    lips_period,lips_shift,MA_method,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[8];
      //--- períodos y desplazamientos de las líneas del Alligator
      pars[0].type=TYPE_INT;
      pars[0].integer_value=jaw_period;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=jaw_shift;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=teeth_period;
      pars[3].type=TYPE_INT;
      pars[3].integer_value=teeth_shift;
      pars[4].type=TYPE_INT;
      pars[4].integer_value=lips_period;
      pars[5].type=TYPE_INT;
      pars[5].integer_value=lips_shift;
      //--- tipo de suavizado
      pars[6].type=TYPE_INT;
      pars[6].integer_value=MA_method;
      //--- tipo de precio
      pars[7].type=TYPE_INT;
      pars[7].integer_value=applied_price;
      //--- crearemos el manejador
      handle=IndicatorCreate(name,period,IND_GATOR,8,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
```

```
      PrintFormat("Fallo al crear el manejador del indicador iGator para el par %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Gator (
   short_name=StringFormat("iGator(%s/%s, %d, %d ,%d, %d, %d, %d)",name,EnumToString(p
                           jaw_period,jaw_shift,teeth_period,teeth_shift,lips_period,
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iGator
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate(
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate(
     {
      //--- si el array GatorUpBuffer supera el número de valores en el indicador iGa
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador(
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
```

```
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador Gator Oscillator
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArraysFromBuffers(GatorUpBuffer,GatorUpColors,GatorDownBuffer,GatorDownCol
      shift,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Gator Oscillator
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iGator        |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &ups_buffer[],         // búfer indicador para el h
                           double &up_color_buffer[],    // búfer indicador para los
                           double &downs_buffer[],       // búfer indicador para el h
                           double &downs_color_buffer[], // búfer indicador para los
                           int u_shift,                  // desplazamiento para el hi
                           int ind_handle,               // manejador del indicador i
                           int amount                    // número de valores a copia
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array GatorUpBuffer con los valores desde el búfer indic
   if(CopyBuffer(ind_handle,0,-u_shift,amount,ups_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iGator, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array GatorUpColors con los valores desde el búfer indic
   if(CopyBuffer(ind_handle,1,-u_shift,amount,up_color_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iGator, código del er
```

```
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array GatorDownBuffer con los valores desde el búfer ind
   if(CopyBuffer(ind_handle,2,-u_shift,amount,downs_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iGator, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array GatorDownColors con los valores desde el búfer ind
   if(CopyBuffer(ind_handle,3,-u_shift,amount,downs_color_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iGator, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iIchimoku

Devuelve el manejador del indicador Ichimoku Kinko Hyo.

```
int  iIchimoku(
   string           symbol,           // nombre del símbolo
   ENUM_TIMEFRAMES  period,           // período
   int              tenkan_sen,       // período Tenkan-sen
   int              kijun_sen,        // período Kijun-sen
   int              senkou_span_b     // período Senkou Span B
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

tenkan_sen

[in] Período promedio Tenkan Sen.

kijun_sen

[in] Período promedio Kijun Sen.

senkou_span_b

[in] Período promedio Senkou Span B.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Números de búfers: 0 - TENKANSEN_LINE, 1 - KIJUNSEN_LINE, 2 - SENKOUSPANA_LINE, 3 - SENKOUSPANB_LINE, 4 - CHINKOUSPAN_LINE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_iIchimoku.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iIchimoku.
```

```
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Ichimoku Kink

#property indicator_chart_window
#property indicator_buffers 5
#property indicator_plots   4
//--- construcción de Tenkan_sen
#property indicator_label1  "Tenkan_sen"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- construcción de Kijun_sen
#property indicator_label2  "Kijun_sen"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrBlue
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- construcción de Senkou_Span
#property indicator_label3  "Senkou Span A;Senkou Span B" // en la Data Window se mos
#property indicator_type3   DRAW_FILLING
#property indicator_color3  clrSandyBrown, clrThistle
#property indicator_style3  STYLE_SOLID
#property indicator_width3  1
//--- construcción de Chinkou_Span
#property indicator_label4  "Chinkou_Span"
#property indicator_type4   DRAW_LINE
#property indicator_color4  clrLime
#property indicator_style4  STYLE_SOLID
#property indicator_width4  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iIchimoku,          // usar iIchimoku
   Call_IndicatorCreate     // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iIchimoku;       // tipo de función
input int                   tenkan_sen=9;              // período de Tenkan-sen
input int                   kijun_sen=26;              // período de Kijun-sen
input int                   senkou_span_b=52;          // período de Senkou Span B
input string                symbol=" ";                // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
```

```
double         Tenkan_sen_Buffer[];
double         Kijun_sen_Buffer[];
double         Senkou_Span_A_Buffer[];
double         Senkou_Span_B_Buffer[];
double         Chinkou_Span_Buffer[];
//--- variable para guardar el manejador del indicador iIchimoku
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Ichimoku Kinko Hyo
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,Tenkan_sen_Buffer,INDICATOR_DATA);
   SetIndexBuffer(1,Kijun_sen_Buffer,INDICATOR_DATA);
   SetIndexBuffer(2,Senkou_Span_A_Buffer,INDICATOR_DATA);
   SetIndexBuffer(3,Senkou_Span_B_Buffer,INDICATOR_DATA);
   SetIndexBuffer(4,Chinkou_Span_Buffer,INDICATOR_DATA);
//--- estableceremos el desplazamiento para el canal Senkou Span a kijun_sen barras e
   PlotIndexSetInteger(2,PLOT_SHIFT,kijun_sen);
//--- no hace falta establecer el desplazamiento para la línea Chinkou Span, porque l
//--- se guardan en el indicador iIchimoku con un desplazamiento ya fijado
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iIchimoku)
      handle=iIchimoku(name,period,tenkan_sen,kijun_sen,senkou_span_b);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[3];
      //--- períodos y desplazamientos de las líneas del Alligator
      pars[0].type=TYPE_INT;
```

```
      pars[0].integer_value=tenkan_sen;
      pars[1].type=TYPE_INT;
      pars[1].integer_value=kijun_sen;
      pars[2].type=TYPE_INT;
      pars[2].integer_value=senkou_span_b;
      //--- crearemos el manejador
      handle=IndicatorCreate(name,period,IND_ICHIMOKU,3,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iIchimoku para el par %s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Ichimo
   short_name=StringFormat("iIchimoku(%s/%s, %d, %d ,%d)",name,EnumToString(period),
                           tenkan_sen,kijun_sen,senkou_span_b);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iIchimoku
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
```

```
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array Tenkan_sen_Buffer supera el número de valores en el indicador
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador Ichimoku Kinko Hyo
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArraysFromBuffers(Tenkan_sen_Buffer,Kijun_sen_Buffer,Senkou_Span_A_Buffer,
      kijun_sen,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Ichimoku Kinko Hyo
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iIchimoku     |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &tenkan_sen_buffer[],     // búfer indicador de la
                           double &kijun_sen_buffer[],      // búfer indicador de la
                           double &senkou_span_A_buffer[],  // búfer indicador de la
                           double &senkou_span_B_buffer[],  // búfer indicador de la
                           double &chinkou_span_buffer[],   // búfer indicador Chinko
                           int senkou_span_shift,           // desplazamiento de la l
                           int ind_handle,                  // manejador del indicado
                           int amount                       // número de valores a co
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array Tenkan_sen_Buffer con los valores desde el búfer i
```

```
   if(CopyBuffer(ind_handle,0,0,amount,tenkan_sen_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("1.Fallo al copiar los datos desde el indicador iGator, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array Kijun_sen_Buffer con los valores desde el búfer in
   if(CopyBuffer(ind_handle,1,0,amount,kijun_sen_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("2.Fallo al copiar los datos desde el indicador iGator, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array Chinkou_Span_Buffer con los valores desde el búfer
//--- si senkou_span_shift>0 la línea se desplaza hacia el futuro a senkou_span_shift
   if(CopyBuffer(ind_handle,2,-senkou_span_shift,amount,senkou_span_A_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("3.Fallo al copiar los datos desde el indicador iGator, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array Senkou_Span_A_Buffer con los valores desde el búfe
//--- si senkou_span_shift>0 la línea se desplaza hacia el futuro a senkou_span_shift
   if(CopyBuffer(ind_handle,3,-senkou_span_shift,amount,senkou_span_B_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("4.Fallo al copiar los datos desde el indicador iGator, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array Senkou_Span_B_Buffer con los valores desde el búfe
//--- cuando copiamos Chinkou Span no hace falta considerar el desplazamiento porque
//--- se guardan en el indicador iIchimoku con un desplazamiento ya fijado
   if(CopyBuffer(ind_handle,4,0,amount,chinkou_span_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("5.Fallo al copiar los datos desde el indicador iGator, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
```

```
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iBWMFI

Devuelve el manejador del indicador Market Facilitation Index. Tiene sólo un búfer.

```
int  iBWMFI(
   string               symbol,              // nombre del símbolo
   ENUM_TIMEFRAMES      period,              // período
   ENUM_APPLIED_VOLUME  applied_volume       // tipo de volumen para el cálculo
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

applied_volume

[in] Volumen usado. Puede ser cualquier valor de la enumeración ENUM_APPLIED_VOLUME.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                 Demo_iBWMFI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iBWMFI."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- construcción de iBWMFI
#property indicator_label1  "iBWMFI"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrLime,clrSaddleBrown,clrBlue,clrPink
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iBWMFI,            // usar iBWMFI
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iBWMFI;          // tipo de función
input ENUM_APPLIED_VOLUME   applied_volume=VOLUME_TICK;// tipo de volumen
input string                symbol=" ";                // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double          iBWMFIBuffer[];
double          iBWMFIColors[];
//--- variable para guardar el manejador del indicador iBWMFI
int     handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Market Facilitation In
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,iBWMFIBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iBWMFIColors,INDICATOR_COLOR_INDEX);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iBWMFI)
      handle=iBWMFI(name,period,applied_volume);
```

```
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      //--- tipo de volumen
      pars[0].type=TYPE_INT;
      pars[0].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_BWMFI,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iIchimoku para el par %s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Market
   short_name=StringFormat("iBWMFI(%s/%s, %s)",name,EnumToString(period),
                           EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iBWMFI
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
```

```
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array Tenkan_sen_Buffer supera el número de valores en el indicador
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador Market Facilitation Inde
//--- si FillArraysFromBuffer ha devuelto false, significa que los datos aún no están
   if(!FillArraysFromBuffers(iBWMFIBuffer,iBWMFIColors,handle,values_to_copy)) return
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Market Facilitation Index
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iBWMFI        |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &values[],     // búfer indicador de valores del his
                           double &colors[],     // búfer indicador de colores del his
                           int ind_handle,       // manejador del indicador iBWMFI
                           int amount            // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iBWMFIBuffer con los valores desde el búfer indica
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iBWMFI, código del er
```

```
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- llenamos una parte del array iBWMFIColors con los valores desde el búfer indica
   if(CopyBuffer(ind_handle,1,0,amount,colors)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iBWMFI, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iMomentum

Devuelve el manejador del indicador Momentum. Tiene sólo un búfer.

```
int  iMomentum(
   string                symbol,            // nombre del símbolo
   ENUM_TIMEFRAMES       period,            // período
   int                   mom_period,        // período promedio
   ENUM_APPLIED_PRICE    applied_price      // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

mom_period

[in] Período promedio (número de barras) para calcular el cambio del precio.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_iMomentum.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iMomentum."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Momentum está

#property indicator_separate_window
#property indicator_buffers 1
```

```
#property indicator_plots   1
//--- plot iMomentum
#property indicator_label1  "iMomentum"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMomentum,          // usar iMomentum
   Call_IndicatorCreate     // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iMomentum;       // tipo de función
input int                   mom_period=14;             // período de momentum
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // tipo de precio
input string                symbol=" ";                // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double          iMomentumBuffer[];
//--- variable para guardar el manejador del indicador iMomentum
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Momentum
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iMomentumBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
```

```
      }
//--- crearemos el manejador del indicador
   if(type==Call_iMomentum)
      handle=iMomentum(name,period,mom_period,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[2];
      //--- período
      pars[0].type=TYPE_INT;
      pars[0].integer_value=mom_period;
      //--- tipo de precio
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_MOMENTUM,2,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iMomentum para el par %s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Moment
   short_name=StringFormat("iMomentum(%s/%s, %d, %s)",name,EnumToString(period),
                           mom_period, EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
```

```
//--- número de valores copiados desde el indicador iMomentum
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata de la primera inicialización de la calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iMomentumBuffer supera el número de valores en el indicador i
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iMomentumBuffer con los valores desde el indicador Momentum
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iMomentumBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Momentum
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iMomentum         |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // búfer indicador de valores Momentum
                         int ind_handle,    // manejador del indicador iMomentum
                         int amount         // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
```

```
   ResetLastError();
//--- llenamos una parte del array iMomentumBuffer con los valores desde el búfer ind
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iMomentum, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iMFI

Cálculo de Money Flow Index.

```
int  iMFI(
   string               symbol,             // nombre del símbolo
   ENUM_TIMEFRAMES      period,             // período
   int                  ma_period,          // período promedio
   ENUM_APPLIED_VOLUME  applied_volume      // tipo de volumen para el cálculo
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio (número de barras) para el cálculo del indicador.

applied_volume

[in] Valor utilizado. Puede ser uno de ENUM_APPLIED_VOLUME.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iMFI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iMFI."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Money Flow In

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
```

```
//--- construcción de iMFI
#property indicator_label1  "iMFI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- niveles horizontales en la ventana del indicador
#property indicator_level1  20
#property indicator_level2  80
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMFI,              // usar iMFI
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iMFI;              // tipo de función
input int                  ma_period=14;                // período
input ENUM_APPLIED_VOLUME  applied_volume=VOLUME_TICK;  // tipo de volumen
input string               symbol=" ";                  // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;       // timeframe
//--- búfer indicador
double         iMFIBuffer[];
//--- variable para guardar el manejador del indicador iMFI
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Money Flow Index
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iMFIBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
```

```
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iMFI)
      handle=iMFI(name,period,ma_period,applied_volume);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[2];
      //--- período
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- tipo de volumen
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_MFI,2,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iMFI para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Money
   short_name=StringFormat("iMFI(%s/%s, %d, %s)",name,EnumToString(period),
                           ma_period, EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iMFI
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iMFIBuffer supera el número de valores en el indicador iMFI s
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iMFIBuffer con los valores desde el indicador Money Flow Inde
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iMFIBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Money Flow Index
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iMFI              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],  // búfer indicador de valores Money Flow
                         int ind_handle,    // manejador del indicador iMFI
                         int amount         // número de valores a copiar
                         )
```

```
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iMFIBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,0,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iMFI, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iMA

Devuelve el manejador del indicador de la media móvil. Tiene sólo un búfer.

```
int  iMA(
   string                symbol,            // nombre del símbolo
   ENUM_TIMEFRAMES       period,            // período
   int                   ma_period,         // período promedio
   int                   ma_shift,          // desplazamiento horizontal del indicador
   ENUM_MA_METHOD        ma_method,         // tipo de suavizado
   ENUM_APPLIED_PRICE    applied_price      // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para calcular la media móvil.

ma_shift

[in] Desplazamiento del indicador con relación al gráfico de precios.

ma_method

[in] Método de promedio. Puede obtener cualquier valor de ENUM_MA_METHOD.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                     Demo_iMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
```

```
#property description "de los búfers indicadores para el indicador técnico iMA."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los de la Moving Aver

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iMA
#property indicator_label1  "iMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMA,               // usar iMA
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iMA;               // tipo de función
input int                  ma_period=10;                // período de la media
input int                  ma_shift=0;                  // desplazamiento
input ENUM_MA_METHOD       ma_method=MODE_SMA;          // tipo de suavizado
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE;   // tipo de precio
input string               symbol=" ";                  // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;       // timeframe
//--- búfer indicador
double         iMABuffer[];
//--- variable para guardar el manejador del indicador iMA
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Moving Average
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iMABuffer,INDICATOR_DATA);
```

```
//--- fijaremos el desplazamiento
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iMA)
      handle=iMA(name,period,ma_period,ma_shift,ma_method,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[4];
      //--- período
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- desplazamiento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- tipo de suavizado
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- tipo de precio
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_MA,4,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iMA para el par %s/%s, c
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Moving
   short_name=StringFormat("iMA(%s/%s, %d, %d, %s, %s)",name,EnumToString(period),
                           ma_period, ma_shift,EnumToString(ma_method),EnumToString(a
```

```
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iMA
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iMABuffer supera el número de valores en el indicador iMA sob
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array iMABuffer con los valores desde el indicador Moving Average
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
```

```
                              short_name,
                              values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Moving Average
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iMA               |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &values[],   // búfer indicador de valores Moving Ave
                         int shift,          // desplazamiento
                         int ind_handle,     // manejador del indicador iMA
                         int amount          // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iMABuffer con los valores desde el búfer indicador
   if(CopyBuffer(ind_handle,0,-shift,amount,values)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iMA, código del error
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iOsMA

Devuelve el manejador del indicador Moving Average of Oscillator. El oscilador OsMA muestra la diferencia entre los valores del MACD y su línea de señales. Tiene sólo un búfer.

```
int  iOsMA(
   string               symbol,              // nombre del símbolo
   ENUM_TIMEFRAMES      period,              // período
   int                  fast_ema_period,     // período de la media móvil rápida
   int                  slow_ema_period,     // período de la media móvil lenta
   int                  signal_period,       // período promedio para su deferencia
   ENUM_APPLIED_PRICE   applied_price        // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

fast_ema_period

[in] Período medio para calcular la media móvil rápida.

slow_ema_period

[in] Período medio para calcular la media móvil lenta.

signal_period

[in] Período medio para calcular la línea de señales.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

En algunos sistemas este oscilador también se conoce como el histograma MACD.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iOsMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
```

```
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iOsMA."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los de la Moving Aver

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iOsMA
#property indicator_label1  "iOsMA"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iOsMA,             // usar iOsMA
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iOsMA;           // tipo de función
input int                  fast_ema_period=12;        // período de la media rápida
input int                  slow_ema_period=26;        // período de la media lenta
input int                  signal_period=9;           // período promedio de la difer
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // tipo de precio
input string               symbol=" ";                // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double         iOsMABuffer[];
//--- variable para guardar el manejador del indicador iAMA
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Moving Average of Osci
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
```

```
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iOsMABuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iOsMA)
      handle=iOsMA(name,period,fast_ema_period,slow_ema_period,signal_period,applied_
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[4];
      //--- período rápido
      pars[0].type=TYPE_INT;
      pars[0].integer_value=fast_ema_period;
      //--- período lento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=slow_ema_period;
      //--- período de promedio de la diferencia entre la media lenta y la rápida
      pars[2].type=TYPE_INT;
      pars[2].integer_value=signal_period;
      //--- tipo de precio
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_OSMA,4,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iOsMA para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Moving
```

```
   short_name=StringFormat("iOsMA(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),
                           fast_ema_period,slow_ema_period,signal_period,EnumToString
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iOsMA
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iOsMABuffer supera el número de valores en el indicador iOsMA
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador iOsMA
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iOsMABuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
```

```
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Moving Average of Oscillator
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iOsMA             |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &ama_buffer[],  // búfer indicador de valores OsMA
                         int ind_handle,        // manejador del indicador iOsMA
                         int amount             // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iOsMABuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,0,0,amount,ama_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iOsMA, código del err
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iMACD

Devuelve el manejador del indicador Moving Averages Convergence/Divergence. En los sistemas, donde OsMA lleva el nombre del Histograma MACD, este indicador se muestra en forma de dos líneas. En el terminal de cliente la convergencia/divergencia de las medias móviles se ve en forma de un histograma.

```
int  iMACD(
   string              symbol,              // nombre del símbolo
   ENUM_TIMEFRAMES     period,              // período
   int                 fast_ema_period,     // período de la media móvil rápida
   int                 slow_ema_period,     // período de la media móvil lenta
   int                 signal_period,       // período promedio para su deferencia
   ENUM_APPLIED_PRICE  applied_price        // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

fast_ema_period

[in] Período medio para calcular la media móvil rápida.

slow_ema_period

[in] Período medio para calcular la media móvil lenta.

signal_period

[in] Período medio para calcular la línea de señales.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Números de búfers: 0 - MAIN_LINE, 1 - SIGNAL_LINE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                  Demo_iMACD.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
```

```
//|                                                 http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iMACD."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del MACD estándar

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
//--- construcción de MACD
#property indicator_label1  "MACD"
#property indicator_type1   DRAW_HISTOGRAM
#property indicator_color1  clrSilver
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- construcción de Signal
#property indicator_label2  "Signal"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_DOT
#property indicator_width2  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iMACD,              // usar iMACD
   Call_IndicatorCreate     // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iMACD;           // tipo de función
input int                  fast_ema_period=12;        // período de la media rápida
input int                  slow_ema_period=26;        // período de la media lenta
input int                  signal_period=9;           // período promedio de la difer
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // tipo de precio
input string               symbol=" ";                // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // período de tiempo
//--- búfers indicadores
double         MACDBuffer[];
double         SignalBuffer[];
//--- variable para guardar el manejador del indicador iMACD
int    handle;
```

```
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Moving Averages Conver
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,MACDBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iMACD)
      handle=iMACD(name,period,fast_ema_period,slow_ema_period,signal_period,applied_
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[4];
      //--- período rápido
      pars[0].type=TYPE_INT;
      pars[0].integer_value=fast_ema_period;
      //--- período lento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=slow_ema_period;
      //--- período de promedio de la diferencia entre la media lenta y la rápida
      pars[2].type=TYPE_INT;
      pars[2].integer_value=signal_period;
      //--- tipo de precio
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_MACD,4,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
```

```
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iMACD para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Moving
   short_name=StringFormat("iMACD(%s/%s,%d,%d,%d,%s)",name,EnumToString(period),
                           fast_ema_period,slow_ema_period,signal_period,EnumToString
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iMACD
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array MACDBuffer supera el número de valores en el indicador iMACD
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
```

```
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador iMACD
//--- si FillArraysFromBuffers ha devuelto false, significa que los datos no están li
   if(!FillArraysFromBuffers(MACDBuffer,SignalBuffer,handle,values_to_copy)) return(0
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Moving Averages Convergence/D
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iMACD         |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &macd_buffer[],    // búfer indicador de valores de
                           double &signal_buffer[],  // búfer indicador de la línea d
                           int ind_handle,           // manejador del indicador iMACD
                           int amount                // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iMACDBuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,0,0,amount,macd_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iMACD, código del err
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }

//--- llenamos una parte del array SignalBuffer con los valores desde el búfer indica
   if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iMACD, código del err
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
```

```
      }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iOBV

Devuelve el manejador del indicador On Balance Volume. Tiene sólo un búfer.

```
int  iOBV(
   string                symbol,             // nombre del símbolo
   ENUM_TIMEFRAMES       period,             // período
   ENUM_APPLIED_VOLUME   applied_volume      // tipo de volumen para el cálculo
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

applied_volume

[in] Volumen usado. Puede ser cualquier valor de la enumeración ENUM_APPLIED_VOLUME.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                    Demo_iOBV.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iOBV."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//---  iOBV
#property indicator_label1  "iOBV"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iOBV ,             // usar iOBV
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iOBV;               // tipo de función
input ENUM_APPLIED_VOLUME   applied_volume=VOLUME_TICK;   // tipo de volumen
input string                symbol=" ";                   // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;        // timeframe
//--- búfers indicadores
double          iOBVBuffer[];
//--- variable para guardar el manejador del indicador iOBV
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador On Balance Volume
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iOBVBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iOBV)
      handle=iOBV(name,period,applied_volume);
   else
     {
```

```
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      //--- tipo de volumen
      pars[0].type=TYPE_INT;
      pars[0].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_OBV,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iOBV para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador On Bal
   short_name=StringFormat("iOBV(%s/%s, %s)",name,EnumToString(period),
                           EnumToString(applied_volume));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iOBV
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
```

```
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iOBVBuffer supera el número de valores en el indicador iOBV s
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador iOBV
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iOBVBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador On Balance Volume
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iOBV              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &obv_buffer[],  // búfer indicador de valores OBV
                         int ind_handle,        // manejador del indicador iOBV
                         int amount             // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iOBVBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,0,amount,obv_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iOBV, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
```

```
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iSAR

Devuelve el manejador del indicador Parabolic Stop and Reverse system. Tiene sólo un búfer.

```
int  iSAR(
   string           symbol,      // nombre del símbolo
   ENUM_TIMEFRAMES  period,      // período
   double           step,        // paso de incremento de velocidad - aceleración
   double           maximum      // coeficiente máximo de seguir un precio
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

step

[in] Aumento del nivel de stop, normalmente - 0.02.

maximum

[in] Nivel máximo de stop, normalmente - 0.2.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iSAR.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iSAR."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Parabolic Sto

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
```

```
//--- construcción de iSAR
#property indicator_label1  "iSAR"
#property indicator_type1   DRAW_ARROW
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iSAR,              // usar iSAR
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iSAR;              // tipo de función
input double               step=0.02;                   // paso - factor de aceleraci
input double               maximum=0.2;                 // valor máximo del paso
input string               symbol=" ";                  // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;       // timeframe
//--- búfers indicadores
double         iSARBuffer[];
//--- variable para guardar el manejador del indicador iSAR
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Parabolic SAR
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iSARBuffer,INDICATOR_DATA);
//--- estableceremos el código del símbolo desde el conjunto Wingdings para la propie
   PlotIndexSetInteger(0,PLOT_ARROW,159);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
```

```
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iSAR)
      handle=iSAR(name,period,step,maximum);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[2];
      //--- valor del paso
      pars[0].type=TYPE_DOUBLE;
      pars[0].double_value=step;
      //--- valor límite del paso que se puede utilizar en el cálculo
      pars[1].type=TYPE_DOUBLE;
      pars[1].double_value=maximum;
      handle=IndicatorCreate(name,period,IND_SAR,2,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iSAR para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Parabo
   short_name=StringFormat("iSAR(%s/%s, %G, %G)",name,EnumToString(period),
                           step,maximum);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
```

```
  {
//--- número de valores copiados desde el indicador iSAR
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iSARBuffer supera el número de valores en el indicador iSAR s
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador iSAR
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iSARBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Parabolic SAR
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iSAR              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &sar_buffer[],  // búfer indicador de valores Parabol
                         int ind_handle,        // manejador del indicador iSAR
                         int amount             // número de valores a copiar
                         )
  {
```

```
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iSARBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,0,amount,sar_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iSAR, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iRSI

Devuelve el manejador del indicador Relative Strength Index. Tiene sólo un búfer.

```
int  iRSI(
   string               symbol,            // nombre del símbolo
   ENUM_TIMEFRAMES      period,            // período
   int                  ma_period,         // período promedio
   ENUM_APPLIED_PRICE   applied_price      // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para calcular el índice.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iRSI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iRSI."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Relative Stre

#property indicator_separate_window
#property indicator_buffers 1
```

```
#property indicator_plots   1
//--- construcción de iRSI
#property indicator_label1  "iRSI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrDodgerBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- límites para visualizar valores en la ventana del indicador
#property indicator_maximum 100
#property indicator_minimum 0
//--- niveles horizontales en la ventana del indicador
#property indicator_level1  70.0
#property indicator_level2  30.0
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iRSI,              // usar iRSI
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iRSI;              // tipo de función
input int                   ma_period=14;                // período promedio
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE;   // tipo de precio
input string                symbol=" ";                  // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;       // timeframe
//--- búfer indicador
double          iRSIBuffer[];
//--- variable para guardar el manejador del indicador iRSI
int     handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Relative Strength Inde
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iRSIBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
```

```
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iRSI)
      handle=iRSI(name,period,ma_period,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[2];
      //--- período de la media móvil
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- valor límite del paso que se puede utilizar en el cálculo
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_RSI,2,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iRSI para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Relati
   short_name=StringFormat("iRSI(%s/%s, %d, %d)",name,EnumToString(period),
                           ma_period,applied_price);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iRSI
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iRSIBuffer supera el número de valores en el indicador iRSI s
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array con los valores desde el indicador iRSI
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iRSIBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Relative Strength Index
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iRSI              |
//+------------------------------------------------------------------+
```

```
bool FillArrayFromBuffer(double &rsi_buffer[],  // búfer indicador de valores Relativ
                         int ind_handle,        // manejador del indicador iSAR
                         int amount             // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iRSIBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,0,amount,rsi_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iRSI, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iRVI

Devuelve el manejador del indicador Relative Vigor Index.

```
int  iRVI(
   string           symbol,        // nombre del símbolo
   ENUM_TIMEFRAMES  period,        // período
   int              ma_period      // período promedio
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para calcular el índice.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Números de búfers: 0 - MAIN_LINE, 1 - SIGNAL_LINE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iRVI.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iRVI."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Relative Vigo

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
```

```
//--- construcción de RVI
#property indicator_label1  "RVI"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- construcción de Signal
#property indicator_label2  "Signal"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iRVI,              // usar iRVI
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iRVI;              // tipo de función
input int                  ma_period=10;                // período para el cálculo
input string               symbol=" ";                  // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;       // timeframe
//--- búfers indicadores
double         RVIBuffer[];
double         SignalBuffer[];
//--- variable para guardar el manejador del indicador iRVI
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Relative Vigor Index
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,RVIBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
```

```
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iRVI)
      handle=iRVI(name,period,ma_period);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      //--- período para el cálculo
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      handle=IndicatorCreate(name,period,IND_RVI,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iRVI para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Relati
   short_name=StringFormat("iRSI(%s/%s, %d, %d)",name,EnumToString(period),ma_period)
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iRVI
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array RVIBuffer supera el número de valores en el indicador iRVI so
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador iRVI
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(RVIBuffer,SignalBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Relative Vigor Index
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iRVI          |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &rvi_buffer[],     // búfer indicador de valores Rela
                         double &signal_buffer[],  // búfer indicador de la línea de
                         int ind_handle,           // manejador del indicador iRVI
                         int amount                // número de valores a copiar
```

```
                          )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iRVIBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,0,amount,rvi_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iRVI, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- llenamos una parte del array SignalBuffer con los valores desde el búfer indica
   if(CopyBuffer(ind_handle,1,0,amount,signal_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iRVI, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iStdDev

Devuelve el manejador del indicador Standard Deviation. Tiene sólo un búfer.

```
int  iStdDev(
   string               symbol,            // nombre del símbolo
   ENUM_TIMEFRAMES      period,            // período
   int                  ma_period,         // período promedio
   int                  ma_shift,          // desplazamiento horizontal del indicador
   ENUM_MA_METHOD       ma_method,         // tipo de suavizado
   ENUM_APPLIED_PRICE   applied_price      // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio para el cálculo del indicador.

ma_shift

[in] Desplazamiento del indicador con relación al gráfico de precios.

ma_method

[in] Método de promedio. Puede ser uno de los valores de ENUM_MA_METHOD.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                Demo_iStdDev.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
```

```
#property description "de los búfers indicadores para el indicador técnico iStdDev."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Standard Devi

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iStdDev
#property indicator_label1  "iStdDev"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrMediumSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iStdDev,           // usar iStdDev
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iStdDev;         // tipo de función
input int                  ma_period=20;              // período promedio
input int                  ma_shift=0;                // desplazamiento
input ENUM_MA_METHOD       ma_method=MODE_SMA;        // tipo de suavizado
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // tipo de precio
input string               symbol=" ";                // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double         iStdDevBuffer[];
//--- variable para guardar el manejador del indicador iStdDev
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Standard Deviation
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iStdDevBuffer,INDICATOR_DATA);
```

```
//--- fijaremos el desplazamiento
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iStdDev)
      handle=iStdDev(name,period,ma_period,ma_shift,ma_method,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[4];
      //--- período
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- desplazamiento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- tipo de suavizado
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_method;
      //--- tipo de precio
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_STDDEV,4,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iStdDev para el par %s/%
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Standa
   short_name=StringFormat("iStdDev(%s/%s, %d, %d, %s, %s)",name,EnumToString(period)
                           ma_period,ma_shift,EnumToString(ma_method),EnumToString(ap
```

```
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iStdDev
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iStdDevBuffer supera el número de valores en el indicador iSt
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array con los valores desde el indicador Standard Deviation
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iStdDevBuffer,ma_shift,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
```

```
                              short_name,
                              values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Standard Deviation
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iStdDev           |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &std_buffer[],  // búfer indicador de la línea Standa
                         int std_shift,         // desplazamiento de la línea Standar
                         int ind_handle,        // manejador del indicador iStdDev
                         int amount             // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iStdDevBuffer con los valores desde el búfer indic
   if(CopyBuffer(ind_handle,0,-std_shift,amount,std_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iStdDev, código del e
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iStochastic

Devuelve el manejador del indicador Stochastic Oscillator.

```
int  iStochastic(
   string           symbol,          // nombre del símbolo
   ENUM_TIMEFRAMES  period,          // período
   int              Kperiod,         // K-período (número de barras para calcular)
   int              Dperiod,         // D-período (período de suavizado inicial)
   int              slowing,         // suavizado final
   ENUM_MA_METHOD   ma_method,       // tipo de suavizado
   ENUM_STO_PRICE   price_field      // modo de cálculo del estocástico
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

Kperiod

[in] K-período (cantidad de barras) para calcular la línea %K.

Dperiod

[in] Período promedio para calcular la línea %D.

slowing

[in] Valor de ralentización.

ma_method

[in] Método de promedio. Puede ser cualquier valor de la enumeración ENUM_MA_METHOD.

price_field

[in] Parámetro de selección de precios para calcular. Puede ser uno de los valores de ENUM_STO_PRICE.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Nota

Números de búfers: 0 - MAIN_LINE, 1 - SIGNAL_LINE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                              Demo_iStochastic.mq5 |
```

```
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iStochasti
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Stochastic Os

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   2
//--- construcción de Stochastic
#property indicator_label1  "Stochastic"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrLightSeaGreen
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- construcción de Signal
#property indicator_label2  "Signal"
#property indicator_type2   DRAW_LINE
#property indicator_color2  clrRed
#property indicator_style2  STYLE_SOLID
#property indicator_width2  1
//--- estableceremos el límite para los valores del indicador
#property indicator_minimum 0
#property indicator_maximum 100
//--- niveles horizontales en la ventana del indicador
#property indicator_level1  -100.0
#property indicator_level2  100.0
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iStochastic,       // usar iStochastic
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iStochastic;     // tipo de función
input int                  Kperiod=5;                 // K-período (número de barras
input int                  Dperiod=3;                 // D-período (período de suaviz
input int                  slowing=3;                 // período para el suavizado fi
input ENUM_MA_METHOD       ma_method=MODE_SMA;        // tipo de suavizado
```

```
input ENUM_STO_PRICE       price_field=STO_LOWHIGH;   // modo de cálculo del estocást
input string               symbol=" ";                // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // período de tiempo
//--- búfers indicadores
double         StochasticBuffer[];
double         SignalBuffer[];
//--- variable para guardar el manejador del indicador iStochastic
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Stochastic Oscillator
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación de los arrays a los búfers indicadores
   SetIndexBuffer(0,StochasticBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,SignalBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iStochastic)
      handle=iStochastic(name,period,Kperiod,Dperiod,slowing,ma_method,price_field);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[5];
      //--- período K para el cálculo
      pars[0].type=TYPE_INT;
      pars[0].integer_value=Kperiod;
      //--- período D para suavizado inicial
      pars[1].type=TYPE_INT;
      pars[1].integer_value=Dperiod;
      //--- período K para el suavizado final
      pars[2].type=TYPE_INT;
```

```
      pars[2].integer_value=slowing;
      //--- tipo de suavizado
      pars[3].type=TYPE_INT;
      pars[3].integer_value=ma_method;
      //--- modo de cálculo del estocástico
      pars[4].type=TYPE_INT;
      pars[4].integer_value=price_field;
      handle=IndicatorCreate(name,period,IND_STOCHASTIC,5,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iStochastic para el par
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Stocha
   short_name=StringFormat("iStochastic(%s/%s, %d, %d, %d, %s, %s)",name,EnumToString
                           Kperiod,Dperiod,slowing,EnumToString(ma_method),EnumToStri
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iStochastic
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
```

```
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array StochasticBuffer supera el número de valores en el indicador
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador iStochastic
//--- si FillArraysFromBuffers ha devuelto false, significa que los datos no están li
   if(!FillArraysFromBuffers(StochasticBuffer,SignalBuffer,handle,values_to_copy)) re
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Stochastic Oscillator
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iStochastic   |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &main_buffer[],    // búfer indicador de valores St
                           double &signal_buffer[],  // búfer indicador de la línea d
                           int ind_handle,           // manejador del indicador iStoc
                           int amount                // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array StochasticBuffer con los valores desde el búfer in
   if(CopyBuffer(ind_handle,MAIN_LINE,0,amount,main_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iStochastic, código d
```

```
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- llenamos una parte del array SignalBuffer con los valores desde el búfer indica
   if(CopyBuffer(ind_handle,SIGNAL_LINE,0,amount,signal_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iStochastic, código d
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iTEMA

Devuelve el manejador del indicador Triple Exponential Moving Average. Tiene sólo un búfer.

```
int  iTEMA(
   string              symbol,           // nombre del símbolo
   ENUM_TIMEFRAMES     period,           // período
   int                 ma_period,        // período promedio
   int                 ma_shift,         // desplazamiento horizontal del indicador
   ENUM_APPLIED_PRICE  applied_price     // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio (cantidad de barras) para el cálculo del indicador.

ma_shift

[in] Desplazamiento del indicador con relación al gráfico de precios.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                  Demo_iTEMA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iTEMA."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
```

```
#property description "Todos los demás parámetros son iguales a los del Triple Expone

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iTEMA
#property indicator_label1  "iTEMA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iTEMA,             // usar iTEMA
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iTEMA;           // tipo de función
input int                   ma_period=14;              // período promedio
input int                   ma_shift=0;                // desplazamiento
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // tipo de precio
input string                symbol=" ";                // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double         iTEMABuffer[];
//--- variable para guardar el manejador del indicador iTEMA
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Triple Exponential Mov
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iTEMABuffer,INDICATOR_DATA);
//--- fijaremos el desplazamiento
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
```

```
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iTEMA)
      handle=iTEMA(name,period,ma_period,ma_shift,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[3];
      //--- período
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- desplazamiento
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ma_shift;
      //--- tipo de precio
      pars[2].type=TYPE_INT;
      pars[2].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_TEMA,3,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iTEMA para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Triple
   short_name=StringFormat("iTEMA(%s/%s, %d, %d, %s)",name,EnumToString(period),
                           ma_period,ma_shift,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iTEMA
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iTEMABuffer supera el número de valores en el indicador iTEMA
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array con los valores desde el indicador Triple Exponential Moving
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iTEMABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Triple Exponential Moving Ave
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
```

```
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iTEMA             |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &tema_buffer[], // búfer indicador de valores Triple
                         int t_shift,           // desplazamiento de la línea
                         int ind_handle,        // manejador del indicador iTEMA
                         int amount             // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iTEMABuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,0,-t_shift,amount,tema_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iTEMA, código del err
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iTriX

Devuelve el manejador del indicador Triple Exponential Moving Averages Oscillator. Tiene sólo un búfer.

```
int  iTriX(
   string              symbol,            // nombre del símbolo
   ENUM_TIMEFRAMES     period,            // período
   int                 ma_period,         // período promedio
   ENUM_APPLIED_PRICE  applied_price      // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

ma_period

[in] Período promedio (cantidad de barras) para el cálculo del indicador.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iTriX.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iTriX."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
```

```
#property indicator_plots   1
//--- construcción de iTriX
#property indicator_label1  "iTriX"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrRed
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iTriX,              // usar iTriX
   Call_IndicatorCreate     // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iTriX;           // tipo de función
input int                   ma_period=14;              // período
input ENUM_APPLIED_PRICE    applied_price=PRICE_CLOSE; // tipo de precio
input string                symbol=" ";                // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double         iTriXBuffer[];
//--- variable para guardar el manejador del indicador iTriX
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Triple Exponential Mov
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iTriXBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
```

```
     }
//--- crearemos el manejador del indicador
   if(type==Call_iTriX)
      handle=iTriX(name,period,ma_period,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[2];
      //--- período
      pars[0].type=TYPE_INT;
      pars[0].integer_value=ma_period;
      //--- tipo de precio
      pars[1].type=TYPE_INT;
      pars[1].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_TRIX,2,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iTriX para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Triple
   short_name=StringFormat("iTriX(%s/%s, %d, %s)",name,EnumToString(period),
                           ma_period,EnumToString(applied_price));
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
```

```
//--- número de valores copiados desde el indicador iTriX
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iTriXBuffer supera el número de valores en el indicador iTriX
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array con los valores desde el indicador Triple Exponential Moving
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iTriXBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Triple Exponential Moving Ave
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iTriX             |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &trix_buffer[], // búfer indicador de valores Triple
                         int ind_handle,        // manejador del indicador iTriX
                         int amount             // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
```

```
   ResetLastError();
//--- llenamos una parte del array iTriXBuffer con los valores desde el búfer indicad
   if(CopyBuffer(ind_handle,0,0,amount,trix_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iTriX, código del err
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iWPR

Devuelve el manejador del indicador Larry Williams' Percent Range. Tiene sólo un búfer.

```
int  iWPR(
   string           symbol,          // nombre del símbolo
   ENUM_TIMEFRAMES  period,          // período
   int              calc_period      // período promedio
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

calc_period

[in] Período promedio (cantidad de barras) para el cálculo del indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                   Demo_iWPR.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iWPR."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iWPR
#property indicator_label1  "iWPR"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrCyan
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//--- estableceremos el límite para los valores del indicador
#property indicator_minimum -100
#property indicator_maximum 0
//--- niveles horizontales en la ventana del indicador
#property indicator_level1  -20.0
#property indicator_level2  -80.0
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iWPR,              // usar iWPR
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iWPR;           // tipo de función
input int                  calc_period=14;           // período
input string               symbol=" ";               // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;    // período de tiempo
//--- búfer indicador
double         iWPRBuffer[];
//--- variable para guardar el manejador del indicador iWPR
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Larry Williams' Percen
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iWPRBuffer,INDICATOR_DATA);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
```

```
     }
//--- crearemos el manejador del indicador
   if(type==Call_iWPR)
      handle=iWPR(name,period,calc_period);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      //--- período
      pars[0].type=TYPE_INT;
      pars[0].integer_value=calc_period;
      handle=IndicatorCreate(name,period,IND_WPR,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iWPR para el par %s/%s,
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Willia
   short_name=StringFormat("iWPR(%s/%s, %d)",name,EnumToString(period),calc_period);
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iWPR
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
```

```
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iWPRBuffer supera el número de valores en el indicador iWPR s
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array con los valores desde el indicador Williams' Percent Range
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iWPRBuffer,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Williams' Percent Range
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iWPR              |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &wpr_buffer[],  // búfer indicador de valores William
                         int ind_handle,        // manejador del indicador iWPR
                         int amount             // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iWPRBuffer con los valores desde el búfer indicado
   if(CopyBuffer(ind_handle,0,0,amount,wpr_buffer)<0)
     {
```

```
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iWPR, código del erro
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iVIDyA

Devuelve el manejador del indicador Variable Index Dynamic Average. Tiene sólo un búfer.

```
int  iVIDyA(
   string               symbol,           // nombre del símbolo
   ENUM_TIMEFRAMES      period,           // período
   int                  cmo_period,       // período Chande Momentum
   int                  ema_period,       // período del factor de suavizado
   int                  ma_shift,         // desplazamiento horizontal del indicador
   ENUM_APPLIED_PRICE   applied_price     // tipo de precio o manejador
   );
```

Parámetros

symbol

[in] Símbolo del instrumento financiero cuyos datos serán usados para calcular el indicador. NULL significa el símbolo actual.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

cmo_period

[in] Período (cantidad de barras) para el cálculo de Chande Momentum Oscillator.

ema_period

[in] Período (cantidad de barras) EMA para el cálculo del factor de suavizado.

ma_shift

[in] Desplazamiento del indicador con relación al gráfico de precios.

applied_price

[in] Precio que se usa. Puede ser cualquier precio de las constantes de precios ENUM_APPLIED_PRICE o un manejador de otro indicador.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                 Demo_iVIDyA.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
```

```
#property description "de los búfers indicadores para el indicador técnico iVIDyA."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e
#property description "Todos los demás parámetros son iguales a los del Variable Inde

#property indicator_chart_window
#property indicator_buffers 1
#property indicator_plots   1
//--- construcción de iVIDyA
#property indicator_label1  "iVIDyA"
#property indicator_type1   DRAW_LINE
#property indicator_color1  clrBlue
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iVIDyA,            // usar iVIDyA
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation             type=Call_iVIDyA;          // tipo de función
input int                  cmo_period=15;             // período Chande Momentum
input int                  ema_period=12;             // período del factor de suaviz
input int                  ma_shift=0;                // desplazamiento
input ENUM_APPLIED_PRICE   applied_price=PRICE_CLOSE; // tipo de precio
input string               symbol=" ";                // símbolo
input ENUM_TIMEFRAMES      period=PERIOD_CURRENT;     // período de tiempo
//--- búfer indicador
double         iVIDyABuffer[];
//--- variable para guardar el manejador del indicador iVIDyA
int    handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Variable Index Dynamic
int    bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array al búfer indicador
   SetIndexBuffer(0,iVIDyABuffer,INDICATOR_DATA);
```

```
//--- fijaremos el desplazamiento
   PlotIndexSetInteger(0,PLOT_SHIFT,ma_shift);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iVIDyA)
      handle=iVIDyA(name,period,cmo_period,ema_period,ma_shift,applied_price);
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[4];
      //--- período Chande Momentum
      pars[0].type=TYPE_INT;
      pars[0].integer_value=cmo_period;
      //--- período del factor de suavizado
      pars[1].type=TYPE_INT;
      pars[1].integer_value=ema_period;
      //--- desplazamiento
      pars[2].type=TYPE_INT;
      pars[2].integer_value=ma_shift;
      //--- tipo de precio
      pars[3].type=TYPE_INT;
      pars[3].integer_value=applied_price;
      handle=IndicatorCreate(name,period,IND_VIDYA,4,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iVIDyA para el par %s/%s
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Triple
   short_name=StringFormat("iVIDyA(%s/%s, %d, %d, %d, %s)",name,EnumToString(period),
                           cmo_period,ema_period,ma_shift,EnumToString(applied_price)
```

```
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iVIDyA
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iWPRBuffer supera el número de valores en el indicador iVIDyA
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos el array con los valores desde el indicador Variable Index Dynamic Ave
//--- si FillArrayFromBuffer ha devuelto false, significa que los datos no están list
   if(!FillArrayFromBuffer(iVIDyABuffer,ma_shift,handle,values_to_copy)) return(0);
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
```

```
                              short_name,
                              values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Variable Index Dynamic Averag
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos el búfer indicador desde el indicador iVIDyA            |
//+------------------------------------------------------------------+
bool FillArrayFromBuffer(double &vidya_buffer[],// búfer indicador de valores Variabl
                         int v_shift,           // desplazamiento de la línea
                         int ind_handle,        // manejador del indicador iVIDyA
                         int amount             // número de valores a copiar
                         )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iVIDyABuffer con los valores desde el búfer indica
   if(CopyBuffer(ind_handle,0,-v_shift,amount,vidya_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iVIDyA, código del er
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

iVolumes

Devuelve el manejador del indicador de volúmenes. Tiene sólo un búfer.

```
int  iVolumes(
   string                symbol,              // nombre del símbolo
   ENUM_TIMEFRAMES       period,              // período
   ENUM_APPLIED_VOLUME   applied_volume       // tipo de volumen para el cálculo
   )
```

Parámetros

symbol

[in] Símbolo del instrumento financiero, cuyos datos serán usados para calcular el indicador. NULL significa el símbolo corriente.

period

[in] Valor del período puede ser uno de los valores de la enumeración ENUM_TIMEFRAMES, 0 significa el timeframe actual.

applied_volume

[in] Volumen usado. Puede ser cualquier valor de la enumeración ENUM_APPLIED_VOLUME.

Valor devuelto

Devuelve el manejador del indicador técnico especificado, en caso del fallo devuelve INVALID_HANDLE.

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                                 Demo_iVolumes.mq5 |
//|                        Copyright 2011, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2011, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"
#property description "El indicador demuestra cómo hay que obtener los datos"
#property description "de los búfers indicadores para el indicador técnico iVolumes."
#property description "El símbolo y el período de tiempo en el que se calcula el indi
#property description "se establecen mediante los parámetros symbol y period."
#property description "El modo de crear el manejador (handle) se establece mediante e

#property indicator_separate_window
#property indicator_buffers 2
#property indicator_plots   1
//--- construcción de iVolumes
#property indicator_label1  "iVolumes"
#property indicator_type1   DRAW_COLOR_HISTOGRAM
#property indicator_color1  clrGreen, clrRed
```

```
#property indicator_style1  STYLE_SOLID
#property indicator_width1  1
//+------------------------------------------------------------------+
//|  Enumeración de modos de crear el manejador                      |
//+------------------------------------------------------------------+
enum Creation
  {
   Call_iVolumes,          // usar iVolumes
   Call_IndicatorCreate    // usar IndicatorCreate
  };
//--- parámetros de entrada
input Creation              type=Call_iVolumes;           // tipo de función
input ENUM_APPLIED_VOLUME   applied_volume=VOLUME_TICK;   // tipo de volumen
input string                symbol=" ";                   // símbolo
input ENUM_TIMEFRAMES       period=PERIOD_CURRENT;        // timeframe
//--- búfers indicadores
double          iVolumesBuffer[];
double          iVolumesColors[];
//--- variable para guardar el manejador del indicador iVolumes
int     handle;
//--- variable para guardar
string name=symbol;
//--- nombre del indicador en el gráfico
string short_name;
//--- vamos a guardar el número de los valores en el indicador Volumes
int     bars_calculated=0;
//+------------------------------------------------------------------+
//| Custom indicator initialization function                         |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- vinculación del array a los búfers indicadores
   SetIndexBuffer(0,iVolumesBuffer,INDICATOR_DATA);
   SetIndexBuffer(1,iVolumesColors,INDICATOR_COLOR_INDEX);
//--- determinamos el símbolo para el que se construye el indicador
   name=symbol;
//--- eliminaremos los espacios del lado izquierdo y derecho
   StringTrimRight(name);
   StringTrimLeft(name);
//--- si después de eso la longitud de la cadena name obtiene el valor cero,
   if(StringLen(name)==0)
     {
      //--- cogeremos el símbolo del gráfico en el que está iniciado el indicador
      name=_Symbol;
     }
//--- crearemos el manejador del indicador
   if(type==Call_iVolumes)
      handle=iVolumes(name,period,applied_volume);
```

```
   else
     {
      //--- llenaremos la estructura con los valores de los parámetros del indicador
      MqlParam pars[1];
      //--- tipo de precio
      pars[0].type=TYPE_INT;
      pars[0].integer_value=applied_volume;
      handle=IndicatorCreate(name,period,IND_VOLUMES,1,pars);
     }
//--- si no se puede crear el manejador
   if(handle==INVALID_HANDLE)
     {
      //--- avisaremos sobre el fallo y mostraremos el número del error
      PrintFormat("Fallo al crear el manejador del indicador iVolumes para el par %s/
                  name,
                  EnumToString(period),
                  GetLastError());
      //--- el trabajo del indicador se finaliza anticipadamente
      return(INIT_FAILED);
     }
//--- mostraremos sobre qué par símbolo/período ha sido calculado el indicador Volume
   short_name=StringFormat("iVolumes(%s/%s, %s)",name,EnumToString(period),EnumToStri
   IndicatorSetString(INDICATOR_SHORTNAME,short_name);
//--- inicialización correcta del indicador
   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[])
  {
//--- número de valores copiados desde el indicador iVolumes
   int values_to_copy;
//--- vamos a averiguar el número de valores calculados en el indicador
   int calculated=BarsCalculated(handle);
   if(calculated<=0)
     {
      PrintFormat("BarsCalculated() ha devuelto %d, el código del error %d",calculate
      return(0);
```

```
     }
//--- si se trata del primer arranque del proceso de calculación de nuestro indicador
//--- o si es necesario calcular el indicador para dos o más barras (entonces algo se
   if(prev_calculated==0 || calculated!=bars_calculated || rates_total>prev_calculate
     {
      //--- si el array iVolumesBuffer supera el número de valores en el indicador iV
      //--- en caso contrario, copiaremos menos que el tamaño de los búfers indicador
      if(calculated>rates_total) values_to_copy=rates_total;
      else                       values_to_copy=calculated;
     }
   else
     {
      //--- significa que no es la primera vez que se calcula nuestro indicador y des
      //--- se ha añadido no más de una barra para la calculación
      values_to_copy=(rates_total-prev_calculated)+1;
     }
//--- llenamos los arrays con los valores desde el indicador iVolumes
//--- si FillArraysFromBuffers ha devuelto false, significa que los datos no están li
   if(!FillArraysFromBuffers(iVolumesBuffer,iVolumesColors,handle,values_to_copy)) re
//--- creamos el mensaje
   string comm=StringFormat("%s ==>  Actualizados los valores en el indicador %s: %d"
                            TimeToString(TimeCurrent(),TIME_DATE|TIME_SECONDS),
                            short_name,
                            values_to_copy);
//--- mostraremos en el gráfico un mensaje de servicio
   Comment(comm);
//--- recordaremos el número de valores en el indicador Volumes
   bars_calculated=calculated;
//--- devolveremos el valor prev_calculated para la siguiente llamada
   return(rates_total);
  }
//+------------------------------------------------------------------+
//| Llenamos los búfers indicadores desde el indicador iVolumes      |
//+------------------------------------------------------------------+
bool FillArraysFromBuffers(double &volume_buffer[],   // búfer indicador de valores V
                           double &color_buffer[],    // búfer indicador de colores
                           int ind_handle,            // manejador del indicador iVol
                           int amount                 // número de valores a copiar
                           )
  {
//--- actualizaremos el código del error
   ResetLastError();
//--- llenamos una parte del array iVolumesBuffer con los valores desde el búfer indi
   if(CopyBuffer(ind_handle,0,0,amount,volume_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iVolumes, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
```

```
      return(false);
     }
//--- llenamos una parte del array iVolumesColors con los valores desde el búfer indi
   if(CopyBuffer(ind_handle,1,0,amount,color_buffer)<0)
     {
      //--- si el proceso de copiado ha fallado, comunicaremos el código del error
      PrintFormat("Fallo al copiar los datos desde el indicador iVolumes, código del
      //--- finalizaremos con el resultado nulo - eso quiere decir que el indicador s
      return(false);
     }
//--- todo ha salido bien
   return(true);
  }
//+------------------------------------------------------------------+
//| Indicator deinitialization function                              |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- limpiaremos el gráfico tras eliminar el indicador
   Comment("");
  }
```

Trabajo con resultados de optimización

Aquí tenemos las funciones que sirven para organizar el procesamiento personalizado de los resultados de optimización en el Probador de estrategias. Se puede llamarlas durante la optimización en los agentes de pruebas, así como de forma local en los EAs y scripts.

Cuando Usted arranca un EA en el Probador de estrategias, puede crear su propio array de datos a base de los tipos simples o estructuras simples (no contienen cadenas, objetos de la clase o objetos del array dinámico). Utilizando la función FrameAdd(), Usted puede guardar este conjunto de datos en una estructura especial que se llama frame (cuadro). Durante la optimización de un EA, cada agente puede enviar al terminal una serie de frames. Todos los frames recibidos, en el orden que vayan llegando de los agentes, se escriben en el archivo *.MQD en la carpeta directorio_del_terminal/MQL5/Files/Tester con el nombre del EA. La llegada de los frames al Terminal de Cliente de parte de un agente de pruebas genera el evento TesterPass.

Los frames se puede almacenar tanto en la memoria del ordenador, como en un archivo con el nombre especificado. Por parte del lenguaje MQL5 no existe limitación alguna respecto al número de los frames.

Función	Acción
FrameFirst	Mueve el puntero de lectura de frames al inicio y reinicia el filtro establecido antes
FrameFilter	Establece el filtro de lectura de frames y mueve el puntero al inicio
FrameNext	Lee el frame y mueve el puntero al siguiente
FrameInputs	Recibe los parámetros input sobre los que está formado el frame
FrameAdd	Añade un frame con datos
ParameterGetRange	Recibe para la variable input la información sobre la banda de valores y el paso de cambios durante la optimización del EA en el Probador de Estrategias
ParameterSetRange	Establece las reglas del uso de la variable input durante la optimización del EA en el Probador de Estrategias: valor, paso de cambio, valor inicial y final

Véase también

Estadística de simulación, Información sobre el programa MQL5 en ejecución

FrameFirst

Mueve el puntero de lectura de frames al inicio y reinicia el filtro establecido.

```
bool  FrameFirst();
```

Valor devuelto

Devuelve true en caso del éxito, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

FrameFilter

Establece el filtro de lectura de frames y mueve el puntero al inicio

```
bool  FrameFilter(
   const string  name,          // nombre público/etiqueta
   long          id             // id público
   );
```

Valor devuelto

Devuelve true en caso del éxito, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Si una cadena vacía se pasa como el primer parámetro, el filtro va a trabajar sólo con el parámetro numérico. Es decir se van a ver todos los frames con id especificado. Si el valor del segundo parámetro es igual a ULONG_MAX, trabaja sólo el filtro de texto.

La llamada a FrameFilter("", ULONG_MAX) equivale a la llamada a FrameFirst(), es decir, equivale a la ausencia del filtro.

FrameNext

Lee el frame actual y mueve el puntero al siguiente. Hay 2 variantes de esta función.

1. Llamar para recibir un valor numérico

```
bool  FrameNext(
   ulong&   pass,      // número del paso en la optimización durante el cual ha sido
   string&  name,      // nombre público/etiqueta
   long&    id,        // id público
   double&  value      // valor
   );
```

2. Llamar para recibir todos los datos del frame

```
bool  FrameNext(
   ulong&   pass,      // número del paso en la optimización durante el cual ha sido
   string&  name,      // nombre público/etiqueta
   long&    id,        // id público
   double&  value,     // valor
   void&    data[]     // array de cualquier tipo
   );
```

Parámetros

pass

[out] Número del paso durante la optimización en el Probador de estrategias.

name

[out] Nombre del identificador.

id

[out] Valor del identificador.

value

[out] Valor numérico individual.

data

[out] Array de cualquier tipo.

Valor devuelto

Devuelve true en caso del éxito, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Si usa la segunda opción de la llamada, necesita procesar de forma correcta los datos recibidos en el array *data[]*.

FrameInputs

Recibe los parámetros input sobre los que está formado el frame con el número de paso especificado.

```
bool  FrameInputs(
   ulong    pass,                // número del paso en la optimización
   string&  parameters[],        // array de cadenas del tipo "parameterN=valueN"
   uint&    parameters_count     // número total de parámetros
   );
```

Parámetros

pass

[out] Número del paso durante la optimización en el Probador de estrategias.

parameters

[out] Array de cadenas con la descripción de los nombres y valores de los parámetros

parameters_count

[out] Número de elementos que contiene el array *parameters[]*.

Valor devuelto

Devuelve true en caso del éxito, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Una vez recibido el número de cadenas *parameters_count* en el array *parameters[]*, se puede organizar el ciclo para el repaso de todas las entradas. Esto permite conocer los valores de los parámetros de entrada del EA para el número establecido del paso.

FrameAdd

Añade un frame con datos. Hay 2 variantes de esta función.

1. Añadir datos desde un archivo

```
bool  FrameAdd(
   const string  name,        // nombre público/etiqueta
   long          id,          // id público
   double        value,       // valor
   const string  filename     // nombre del archivo con datos
   );
```

2. Añadir datos desde un array de cualquier tipo

```
bool  FrameAdd(
   const string  name,        // nombre público/etiqueta
   long          id,          // id público
   double        value,       // valor
   const void&   data[]       // array de cualquier tipo
   );
```

Parámetros

name

[in] Etiqueta pública frame. Se puede utilizarla para el filtro en la función FrameFilter().

id

[in] Identificador público del frame. Se puede utilizarlo para el filtro en la función FrameFilter().

value

[in] Valor numérico para escribir en el frame. Sirve para transmitir un solitario resultado del paso como en la función OnTester().

filename

[in] Nombre del archivo que contiene los datos para agregar al frame. El archivo debe ubicarse en la carpeta MQL5/Files.

data

[in] Array de cualquier tipo para la escritura en el frame. Se pasa por referencia.

Valor devuelto

Devuelve true en caso del éxito, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

ParameterGetRange

Recibe para la variable input la información sobre la banda de valores y el paso de cambios durante la optimización del EA en el Probador de Estrategias. Hay 2 variantes de esta función.

1. Obtención de la información para el parámetro input del tipo entero

```
bool  ParameterGetRange(
   const string  name,           // nombre del parámetro (variable input)
   bool&         enable,         // permitida la optimización del parámetro
   long&         value,          // valor del parámetro
   long&         start,          // valor inicial
   long&         step,           // paso de cambio
   long&         stop            // valor final
   );
```

2. Obtención de la información para el parámetro input del tipo real

```
bool  ParameterGetRange(
   const string  name,           // nombre del parámetro (variable input)
   double&       enable,         // permitida la optimización del parámetro
   double&       value,          // valor del parámetro
   double&       start,          // valor inicial
   double&       step,           // paso de cambio
   double&       stop            // valor final
   );
```

Parámetros

name

[in] Identificador de la variable input. Estas variables son parámetros externos del programa cuyos valores pueden ser establecidos durante el arranque en el gráfico o bien durante la simulación.

enable

[out] Quiere decir que este parámetro se puede utilizar para el repaso de valores en el proceso de optimización en el Probador de Estrategias.

value

[out] Valor del parámetro.

start

[out] Valor inicial del parámetro durante la optimización.

step

[out] Paso de cambio del parámetro durante el repaso de sus valores.

stop

[out] Valor final del parámetro durante la optimización.

Valor devuelto

Devuelve true en caso de la ejecución con éxito, de lo contrario - false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Esta función puede ser invocada sólo desde los manejadores OnTesterInit(), OnTesterPass() y OnTesterDeinit(). Sirve para obtener el valor y el rango de cambio de los parámetros de entrada del EA durante el proceso de optimización en el Probador de Estrategias.

Cuando se llama en OnTesterInit(), la información obtenida se puede utilizar para redefinir las reglas de repaso de cualquier variable input a través de la función ParameterSetRange(). De esta manera, se puede establecer nuevos valores Start, Stop, Step, e incluso excluir completamente este parámetro de la optimización a pesar de los ajustes en el probador. Esto permite crear sus propios guiones de manejo del área de parámetros de entrada durante la optimización. Es decir, excluir de la optimización unos parámetros en función de los valores de los parámetros claves del EA.

Ejemplo:

```
#property description "Asesor Experto para demostrar la función ParameterGetRange()."
#property description "Hay que iniciar en el Probador de Estrategias en el modo de op
//--- input parameters
input int                  Input1=1;
input double               Input2=2.0;
input bool                 Input3=false;
input ENUM_DAY_OF_WEEK     Input4=SUNDAY;

//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- EA sirve sólo para trabajar en el probador
   if(!MQL5InfoInteger(MQL5_OPTIMIZATION))
     {
      MessageBox("¡Hay que iniciar en el Probador de Estrategias en el modo de optimi
      //--- finalizamos anticipadamente el trabajo del EA y lo quitamos del gráfico
      return(INIT_FAILED);
     }
//--- inicialización finalizada con éxito
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| TesterInit function                                              |
//+------------------------------------------------------------------+
void OnTesterInit()
  {
//--- ejemplo para el parámetro input del tipo long
   string name="Input1";
   bool enable;
   long par1,par1_start,par1_step,par1_stop;
   ParameterGetRange(name,enable,par1,par1_start,par1_step,par1_stop);
   Print("El primer parámetro");
   PrintFormat("%s=%d  enable=%s  from %d to %d with step=%d",
               name,par1,(string)enable,par1_start,par1_stop,par1_step);
//--- ejemplo para el parámetro input del tipo double
   name="Input2";
   double par2,par2_start,par2_step,par2_stop;
   ParameterGetRange(name,enable,par2,par2_start,par2_step,par2_stop);
   Print("El segundo parámetro");
   PrintFormat("%s=%G  enable=%s  from %G to %G with step=%G",
               name,par2,(string)enable,par2_start,par2_stop,par2_step);

//--- ejemplo para el parámetro input del tipo bool
   name="Input3";
   long par3,par3_start,par3_step,par3_stop;
   ParameterGetRange(name,enable,par3,par3_start,par3_step,par3_stop);
   Print("El tercer parámetro");
   PrintFormat("%s=%s  enable=%s  from %s to %s",
               name,(string)par3,(string)enable,
               (string)par3_start,(string)par3_stop);
//--- ejemplo para el parámetro input del tipo enumeración
   name="Input4";
   long par4,par4_start,par4_step,par4_stop;
   ParameterGetRange(name,enable,par4,par4_start,par4_step,par4_stop);
   Print("El cuarto parámetro");
   PrintFormat("%s=%s  enable=%s  from %s to %s",
               name,EnumToString((ENUM_DAY_OF_WEEK)par4),(string)enable,
               EnumToString((ENUM_DAY_OF_WEEK)par4_start),
               EnumToString((ENUM_DAY_OF_WEEK)par4_stop));
```

```
  }
//+------------------------------------------------------------------+
//| TesterDeinit function                                            |
//+------------------------------------------------------------------+
void OnTesterDeinit()
  {
//--- este mensaje se mostrará una vez finalizada la optimización
   Print(__FUNCTION__," Optimisation completed");
  }
```

ParameterSetRange

Establece las reglas del uso de la variable input durante la optimización del EA en el Probador de Estrategias: valor, paso de cambio, valor inicial y final. Hay 2 variantes de esta función.

1. Establecer los valores para el parámetro input del tipo entero

```
bool  ParameterSetRange(
   const string  name,          // nombre del parámetro (variable input)
   bool          enable,        // permitir la optimización del parámetro
   long          value,         // valor del parámetro
   long          start,         // valor inicial
   long          step,          // paso de cambio
   long          stop           // valor final
   );
```

2. Establecer los valores para el parámetro input del tipo real

```
bool  ParameterSetRange(
   const string  name,          // nombre del parámetro (variable input)
   double        enable,        // permitir la optimización del parámetro
   double        value,         // valor del parámetro
   double        start,         // valor inicial
   double        step,          // paso de cambio
   double        stop           // valor final
   );
```

Parámetros

name

[in] Identificador de la variable input o sinput. Estas variables son parámetros externos del programa cuyos valores pueden ser establecidos durante el arranque.

enable

[in] Permitir este parámetro para el repaso de valores durante el proceso de optimización en el Probador de Estrategias.

value

[in] Valor del parámetro.

start

[in] Valor inicial del parámetro durante la optimización.

step

[in] Paso de cambio del parámetro durante el repaso de sus valores.

stop

[in] Valor final del parámetro durante la optimización.

Valor devuelto

Devuelve true en caso de la ejecución con éxito, de lo contrario - false. Para obtener la información

sobre el error, hay que llamar a la función GetLastError().

Nota

Esta función puede ser invocada sólo desde el manejador OnTesterInit() durante el inicio de optimización en el Probador de Estrategias. Sirve para establecer el rango y paso de cambio del parámetro, pudiendo también excluir completamente este parámetro del proceso de optimización a pesar de los ajustes en el probador. Además, permite utilizar en la optimización incluso las variables declaradas con el modificador sinput.

La función ParameterSetRange() permite crear sus propios guiones de optimización del EA en el probador en función de los valores de los parámetros claves. Es decir, incluir o excluir de la optimización los parámetros de entrada necesarios, así como establecer el rango y paso de cambio necesarios.

Trabajo con eventos

Este grupo contiene las funciones para trabajar con los eventos personalizados y con los eventos de temporizador. Además de estas funciones, existen también funciones especiales para manejar los eventos predefinidos.

Función	Acción
EventSetMillisecondTimer	Arranca el generador de eventos del temporizador de alta precisión con el período inferior a 1 segundo para el gráfico actual
EventSetTimer	Arranca el generador de eventos de temporizador con la periodicidad especificada para el gráfico actual
EventKillTimer	Detiene la generación de eventos en el gráfico actual según el temporizador
EventChartCustom	Genera los eventos personalizados para el gráfico especificado

Véase también

Tipos de eventos del gráfico

EventSetMillisecondTimer

Esta función indica al Terminal de Cliente que los eventos del temporizador para este EA o indicador deben generarse con una periodicidad inferior a un segundo.

```
bool  EventSetMillisecondTimer(
   int  milliseconds      // cantidad de milisegundos
   );
```

Parámetros

milliseconds

[in] Cantidad de milisegundos que determina la frecuencia de los eventos del temporizador.

Valor devuelto

En caso de la ejecución exitosa devuelve true, de lo contrario false. Para obtener el código del error, hay que llamar a la función GetLastError().

Nota

Esta función sirve para los casos cuando se requiere un temporizador de alta resolución. Es decir, cuando es necesario obtener los eventos del temporizador con más frecuencia que una vez al segundo. Si un temporizador con el período más de 1 segundo es suficiente para Ustes, utilice la función EventSetTimer().

En el Probador de Estrategias se utiliza un intervalo mínimo de 1000 milisegundos. En general, con la disminución del período del temporizador se aumenta el tiempo de simulación, puesto que sube el número de llamadas al manejador de eventos del temporizador. Cuando se trabaja en tiempo real, los eventos del temporizador se generan con una frecuencia no más de 1 vez en cada 10-16 milisegundos, esto está relacionado con las limitaciones de hardware.

Por lo común, esta función debe llamarse desde la función OnInit() o en el constructor de la clase. Un EA o un indicador tiene que tener la función OnTimer() para poder manejar los eventos que llegan desde el temporizador.

Cada EA y cada indicador trabaja con su propio temporizador y recibe los eventos sólo de él. Cuando un programa mql5 finaliza su trabajo, el temporizador se destruye de manera forzada en caso si ha sido creado pero no ha sido desactivado por la función EventKillTimer().

Se puede iniciar sólo un temporizador para cada programa. Cada programa mql5 y cada gráfico tienen su propia cola de eventos en la que se colocan todos los eventos según vayan llegando. Si en la cola ya hay un evento Timer, o este evento se encuentra en el proceso de procesamiento, entonces el nuevo evento Timer no se coloca en la cola del programa mql5.

EventSetTimer

Esta función indica al terminal de cliente que hay que generar los eventos desde el temporizador con la periodicidad especificada para este Asesor Experto o el indicador.

```
bool  EventSetTimer(
   int  seconds      // cantidad de segundos
   );
```

Parámetros

seconds

[in] Cantidad de segundos que determina la frecuencia de aparición de eventos desde el temporizador.

Valor devuelto

En caso de éxito devuelve true, de lo contrario devuelve false. Para obtener el código de error hay que llamar a la función GetLastError().

Nota

Habitualmente esta función debe invocarse desde la función OnInit() o en un constructor de clase. Para manejar los eventos que vienen del temporizador, el Asesor Experto o el indicador debe tener la función OnTimer().

Cada Asesor Experto, igual que cada indicador, trabaja con su propio temporizador y recibe eventos únicamente de él. Una vez finalizado el trabajo de un programa mql5, el temporizador se elimina forzadamente, si ha sido creado pero no ha sido desactivado por la función EventKillTimer().

Para cada programa se puede arrancar no más de un temporizador. Cada programa mql5 y cada gráfico tiene su propia cola de eventos en la que se ponen todos los eventos recién llegados. Si en la cola ya hay un evento Timer, o este evento se encuentra en el proceso de tramitación, entonces el nuevo evento Timer no se coloca en la cola del programa mql5.

EventKillTimer

Esta función indica al terminal de cliente que hace falta detener la generación de eventos desde el temporizador para este Asesor Experto o el indicador.

```
void  EventKillTimer();
```

Valor devuelto

No hay valor devuelto.

Nota

Normalmente esta función debe invocarse desde la función OnDeinit() en el caso si en la función OnInit() ha sido invocada la función EventSetTimer(). O debe invocarse del destructor de clase, si en el constructor de esta clase se llama a la función EventSetTimer().

Cada Asesor Experto y cada indicador trabaja con su propio temporizador y recibe eventos únicamente de él. Una vez finalizado el trabajo de un programa mql5, el temporizador se elimina forzadamente, si ha sido creado pero no ha sido desactivado por la función EventKillTimer().

EventChartCustom

Genera un evento personalizado para el gráfico especificado.

```
bool  EventChartCustom(
   long    chart_id,            // identificador del gráfico que recibe el evento
   ushort  custom_event_id,     // identificador del evento
   long    lparam,              // parámetro del tipo long
   double  dparam,              // parámetro del tipo double
   string  sparam               // parámetro literal del evento
   );
```

Parámetros

chart_id

[in] identificador del gráfico. 0 significa el gráfico actual.

custom_event_id

[in] Identificador del evento personalizado. Este identificador se añade automáticamente al valor CHARTEVENT_CUSTOM y se convierte al tipo entero.

lparam

[in] Parámetro del evento del tipo long que se pasa a la función OnChartEvent.

dparam

[in] Parámetro del evento del tipo double que se pasa a la función OnChartEvent.

sparam

[in] Parámetro del evento del tipo string que se pasa a la función OnChartEvent. Si la cadena tiene más de 63 caracteres, esta cadena se recorta.

Valor devuelto

Devuelve true si un evento personalizado ha sido colocado con éxito a la cola de los eventos del gráfico-receptor del evento. En caso del error, devuelve false. Para obtener el código del error, utilice la función GetLastError().

Nota

El Asesor Experto o indicador que está adjuntado al gráfico procesa este evento utilizando la función OnChartEvent(int event_id, long& lparam, double& dparam, string& sparam).

Para cada tipo del evento, los parámetros de entrada de la función OnChartEvent() tienen determinados valores que son necesarios para procesar este evento. En la tabla de abajo están especificados los eventos y valores que se pasan a través de los parámetros.

Evento	Valor del parámetro id	Valor del parámetro lparam	Valor del parámetro dparam	Valor del parámetro sparam
Evento del teclado	CHARTEVENT_KEYDOWN	código de la tecla pulsada	Número de pulsaciones de la tecla generadas	Valor literal de la máscara de bits que describe el

			mientras ésta se mantenía en estado pulsado	estatus de las teclas del teclado
Eventos del ratón (si para el gráfico está establecida la propiedad CHART_EVENT_MOUSE_MOVE=true)	CHARTEVENT_MOUSE_MOVE	coordinada X	coordinada Y	Valor literal de la máscara de bits que describe el estatus de los botones del ratón
Evento de creación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_CREATE=true)	CHARTEVENT_OBJECT_CREATE	—	—	Nombre del objeto gráfico creado
Evento del cambio de propiedades de un objeto a través del diálogo de propiedades	CHARTEVENT_OBJECT_CHANGE	—	—	Nombre del objeto gráfico modificado
Evento de eliminación de un objeto gráfico (si para el gráfico está establecida la propiedad CHART_EVENT_OBJECT_DELETE=true)	CHARTEVENT_OBJECT_DELETE	—	—	Nombre del objeto gráfico eliminado
Evento de cliquear sobre un gráfico	CHARTEVENT_CLICK	coordinada X	coordinada Y	—
Evento de cliquear sobre un objeto gráfico	CHARTEVENT_OBJECT_CLICK	coordinada X	coordinada Y	Nombre del objeto gráfico en el que ha ocurrido un evento

Evento de mover un objeto gráfico usando el ratón	CHARTEVENT_OBJECT_DRAG	—	—	Nombre del objeto gráfico movido
Evento del fin de edición del texto en el campo de introducción del objeto gráfico "Campo de texto"	CHARTEVENT_OBJECT_ENDEDIT	—	—	Nombre del objeto gráfico "Campo de texto" donde ha finalizado la introducción del texto
Evento de modificación del gráfico	CHARTEVENT_CHART_CHANGE	—	—	—
Identificador del evento de usuario	CHARTEVENT_CUSTOM+N	Valor determinado por la función EventChartCustom()	Valor determinado por la función EventChartCustom()	Valor determinado por la función EventChartCustom()

Ejemplo:

```
//+------------------------------------------------------------------+
//|                                              ButtonClickExpert.mq5 |
//|                        Copyright 2009, MetaQuotes Software Corp. |
//|                                              http://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "2009, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
#property version   "1.00"

string buttonID="Button";
string labelID="Info";
int broadcastEventID=5000;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
//--- creamos un botón para pasar los eventos de usuario
   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,"Botón");
   ObjectSetInteger(0,buttonID,OBJPROP_FONTSIZE,10);
   ObjectSetInteger(0,buttonID,OBJPROP_SELECTABLE,0);

//--- creamos una etiqueta para visualizar la información
   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 hay información");
   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)
  {
//--- comprobamos el evento de apretar el botón del ratón
   if(id==CHARTEVENT_OBJECT_CLICK)
     {
      string clickedChartObject=sparam;
      //--- si hacemos clic en el objeto con el nombre buttonID
      if(clickedChartObject==buttonID)
        {
         //--- estado del botón - está apretado o no
         bool selected=ObjectGetInteger(0,buttonID,OBJPROP_STATE);
         //--- registramos el mensaje de la depuración
         Print("Botón apretado = ",selected);
         int customEventID; // número del evento de usuario a enviar
         string message;    // mensaje a enviar en el evento
         //--- si el botón está apretado
         if(selected)
           {
            message="Botón apretado";
            customEventID=CHARTEVENT_CUSTOM+1;
           }
         else // botón no está apretado
           {
            message="Botón suelto";
            customEventID=CHARTEVENT_CUSTOM+999;
           }
         //--- enviamos un evento de usuario a "nuestro" gráfico
         EventChartCustom(0,customEventID-CHARTEVENT_CUSTOM,0,0,message);
         //--- enviamos el mensaje a todos los gráficos abiertos
         BroadcastEvent(ChartID(),0,"Broadcast Message");
         //--- mensaje de reparación de errores
         Print("Evento enviado con ID = ",customEventID);
        }
      ChartRedraw();// volvemos a dibujar por vía forzada todos los objetos del gráfi
     }

//--- comprobamos un evento a su pertenencia a eventos de usuario
   if(id>CHARTEVENT_CUSTOM)
     {
      if(id==broadcastEventID)
        {
         Print("Recibido el mensaje de difusión del gráfico con id = "+lparam);
        }
      else
        {
         //--- vamos a leer un mensaje de texto en el evento
         string info=sparam;
         Print("Se procesa el evento de USUARIO con ID = ",id);
         //--- mostramos el mensaje en una etiqueta
         ObjectSetString(0,labelID,OBJPROP_TEXT,sparam);
         ChartRedraw();// volvemos a dibujar por vía forzada todos los objetos del gr
        }
     }
  }
//+------------------------------------------------------------------+
//| enviar un mensaje de difusión a todos los gráficos abiertos      |
//+------------------------------------------------------------------+
void BroadcastEvent(long lparam,double dparam,string sparam)
```

```
  {
   int eventID=broadcastEventID-CHARTEVENT_CUSTOM;
   long currChart=ChartFirst();
   int i=0;
   while(i<CHARTS_MAX)           // seguramente tenemos no más de CHARTS_MAX de gráfic
     {
      EventChartCustom(currChart,eventID,lparam,dparam,sparam);
      currChart=ChartNext(currChart); // a base del anterior obtenemos un gráfico nue
      if(currChart==-1) break;        // llegamos al final de la lista de gráficos
      i++;                            // No olvidemos a aumentar el contador
     }
  }
//+------------------------------------------------------------------+
```

Véase también

Eventos de terminal de cliente, Funciones de procesamiento de eventos

Trabajo con OpenCL

Los programas en OpenCLsirven para ejecutar los cálculos en las tarjetas gráficas con el soporte del estándar OpenCL 1.1 o superior. Las tarjetas gráficas modernas cuentan con centenares de pequeños procesadores especializados que son capaces de realizar simultáneamente las operaciones matemáticas sencillas con los flujos de datos entrantes. El lenguaje OpenCL se encarga de estos cálculos paralelos y permite alcanzar una aceleración increíble para algunos tipos de tareas.

Funciones para ejecutar programas en OpenCL:

Función	Acción
CLHandleType	Devuelve el tipo de manejador OpenCL como un valor de la enumeración ENUM_OPENCL_HANDLE_TYPE
CLGetInfoInteger	Devuelve el valor de una propiedad de números enteros para el objeto o dispositivo OpenCL
CLContextCreate	Crea el contexto OpenCL
CLContextFree	Elimina el contexto OpenCL
CLProgramCreate	Crea un programa OpenCL desde el código fuente
CLProgramFree	Elimina un programa OpenCL
CLKernelCreate	Crea la función del arranque OpenCL
CLKernelFree	Elimina la función del arranque OpenCL
CLSetKernelArg	Establece un parámetro para la función OpenCL
CLSetKernelArgMem	Establece el búfer OpenCL como el parámetro de la función OpenCL
CLBufferCreate	Crea el búfer OpenCL
CLBufferFree	Elimina el búfer OpenCL
CLBufferWrite	Escribe un array en el búfer OpenCL
CLBufferRead	Lee el búfer OpenCL en un array
CLExecute	Ejecuta un programa OpenCL

CLHandleType

Devuelve el tipo de manejador OpenCL como un valor de la enumeración ENUM_OPENCL_HANDLE_TYPE.

```
ENUM_OPENCL_HANDLE_TYPE  CLHandleType(
   int  handle     // manejador del objeto OpenCL
   );
```

Parámetros

handle

[in] Manejador para el objeto OpenCL: contexto, kernel, búfer o un programa OpenCL.

Valor devuelto

Tipo del manejador OpenCL como un valor de la enumeración ENUM_OPENCL_HANDLE_TYPE.

ENUM_OPENCL_HANDLE_TYPE

Identificador	Descripción
OPENCL_INVALID	Manejador incorrecto
OPENCL_CONTEXT	Manejador del contexto OpenCL
OPENCL_PROGRAM	Manejador del programa OpenCL
OPENCL_KERNEL	Manejador del kernel OpenCL
OPENCL_BUFFER	Manejador del búfer OpenCL

CLGetInfoInteger

Devuelve el valor de una propiedad de números enteros para el objeto o dispositivo OpenCL.

```
long  CLGetInfoInteger(
   int  handle,                          // manejador del objeto OpenCL o el número
   ENUM_OPENCL_PROPERTY_INTEGER  prop    // propiedad solicitada
   );
```

Parámetros

handle

[in] Manejador para el objeto OpenCL o el número del dispositivo OpenCL. La numeración de los dispositivos OpenCL se empieza desde cero

prop

[in] Tipo de la propiedad solicitada desde la enumeración ENUM_OPENCL_PROPERTY_INTEGER cuyo valor hay que recibir.

Valor devuelto

Valor de la propiedad especificada en caso del éxito, o -1 en caso del error. Para obtener la información sobre el error, utilice la función GetLastError().

ENUM_OPENCL_PROPERTY_INTEGER

Identificador	Descripción	Tipo
CL_DEVICE_COUNT	El número de dispositivos con soporte de OpenCL. Para esta propiedad no hace falta indicar el primer parámetro, es decir, se puede pasar el valor cero para el parámetro *handle*.	int
CL_DEVICE_TYPE	Tipo del dispositivo	ENUM_CL_DEVICE_TYPE
CL_DEVICE_VENDOR_ID	Identificador único del fabricante	uint
CL_DEVICE_MAX_COMPUTE_UNITS	Número de tareas paralelas calculadas en el dispositivo OpenCL. Un grupo de trabajo se encarga de una tarea computacional. El valor mínimo es igual a 1	uint
CL_DEVICE_MAX_CLOCK_FREQUENCY	Frecuencia máxima establecida del dispositivo en MHz.	uint
CL_DEVICE_GLOBAL_MEM_SIZE	Tamaño de la memoria global del dispositivo en bytes	ulong

CL_DEVICE_LOCAL_MEM_SIZE	Tamaño de la memoria local de datos procesados (escenarios) en bytes	uint

La enumeración ENUM_CL_DEVICE_TYPE contiene los posibles tipos de dispositivos con el soporte de OpenCL. Puede obtener el tipo del dispositivo según su número o el handle del objeto OpenCL llamando a CLGetInfoInteger(handle_or_deviceN, CL_DEVICE_TYPE).

ENUM_CL_DEVICE_TYPE

Identificador	Descripción
CL_DEVICE_ACCELERATOR	Acelerador especializado OpenCL (por ejemplo, IBM CELL Blade).
CL_DEVICE_CPU	Uso de la CPU del ordenador como un dispositivo OpenCL. La CPU puede tener uno o más núcleos de cómputo.
CL_DEVICE_GPU	Un dispositivo OpenCL a base de una tarjeta de vídeo.
CL_DEVICE_DEFAULT	Dispositivo OpenCL por defecto. Un dispositivo CL_DEVICE_TYPE_CUSTOM no puede ser el dispositivo predefinido.
CL_DEVICE_CUSTOM	Los aceleradores especializados que no soportan los programas en OpenCL C.

Ejemplo:

```
void OnStart()
  {
   int cl_ctx;
//--- inicialización del contexto OpenCL
   if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
     {
      Print("OpenCL not found");
      return;
     }
//--- Visualizamos la información general sobre el dispositivo OpenCL
   Print("OpenCL type: ",EnumToString((ENUM_CL_DEVICE_TYPE)CLGetInfoInteger(cl_ctx,CL_
   Print("OpenCL vendor ID: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_VENDOR_ID));
   Print("OpenCL units: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_COMPUTE_UNITS));
   Print("OpenCL freq: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_CLOCK_FREQUENCY),"  MHz
   Print("OpenCL global mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_GLOBAL_MEM_SIZE),"  b
   Print("OpenCL local mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_LOCAL_MEM_SIZE),"  byt
//---
  }
```

CLGetInfoString

Esta función devuelve el valor literal de la propiedad para un objeto o dispositivo OpenCL.

```
bool  CLGetInfoString(
   int  handle,                          // manejador del objeto OpenCL o el número
   ENUM_OPENCL_PROPERTY_STRING  prop,    // propiedad solicitada
   string&  value                        // cadena por referencia
   );
```

Parámetros

handle

[in] Manejador para un objeto OpenCL o número del dispositivo OpenCL. La numeración de los dispositivos OpenCL se empieza desde cero.

prop

[in] Tipo de la propiedad solicitada desde la enumeración ENUM_OPENCL_PROPERTY_STRING cuyo valor hay que obtener.

value

[out] Cadena para obtener el valor de la propiedad.

Valor devuelto

true para la ejecución con éxito, false en caso del error. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

ENUM_OPENCL_PROPERTY_STRING

Identificador	Descripción
CL_PLATFORM_PROFILE	CL_PLATFORM_PROFILE - Perfil OpenCL. El nombre del perfil puede ser uno de los siguientes valores: • FULL_PROFILE - la implementación soporta OpenCL (la funcionalidad está definida como una parte de la especificación del núcleo y no requiere las extensiones adicionales para el soporte de OpenCL); • EMBEDDED_PROFILE - la implementación soporta OpenCL como un suplemento. El perfil enmendado se define como un subconjunto para cada versión OpenCL.
CL_PLATFORM_VERSION	Versión OpenCL
CL_PLATFORM_VENDOR	Nombre del fabricante del dispositivo
CL_PLATFORM_EXTENSIONS	Lista de extensiones que soporta la plataforma. Todos los dispositivos relacionados con esta plataforma deben soportar los nombres de las extensiones
CL_DEVICE_NAME	Nombre del dispositivo

CL_DEVICE_VENDOR	Nombre del fabricante
CL_DRIVER_VERSION	Versión del controlador OpenCL en el formato major_number.minor_number
CL_DEVICE_PROFILE	Perfil del dispositivo OpenCL. El nombre del perfil puede ser uno de los siguientes valores: • FULL_PROFILE - la implementación soporta OpenCL (la funcionalidad está definida como una parte de la especificación del núcleo y no requiere las extensiones adicionales para el soporte de OpenCL); • EMBEDDED_PROFILE - la implementación soporta OpenCL como un suplemento. El perfil enmendado se define como un subconjunto para cada versión OpenCL.
CL_DEVICE_VERSION	Versión OpenCL en el formato "OpenCL<space><major_version.minor_version><space><vendor-specific information>"
CL_DEVICE_EXTENSIONS	Lista de extensiones que soporta el dispositivo. La lista puede incluir extensiones soportadas por el fabricante y contener uno o más nombres aprobados: cl_khr_int64_base_atomics cl_khr_int64_extended_atomics cl_khr_fp16 cl_khr_gl_sharing cl_khr_gl_event cl_khr_d3d10_sharing cl_khr_dx9_media_sharing cl_khr_d3d11_sharing
CL_DEVICE_BUILT_IN_KERNELS	Lista de kernels que soporta el dispositivo separados por ";" .
CL_DEVICE_OPENCL_C_VERSION	La versión máxima que soporta el compilador para este dispositivo. Formato de la versión: "OpenCL<space>C<space><major_version.minor_version><space><vendor-specific information> "

Ejemplo:

```
void OnStart()
  {
   int cl_ctx;
   string str;
//--- inicialización del contexto OpenCL
   if((cl_ctx=CLContextCreate(CL_USE_GPU_ONLY))==INVALID_HANDLE)
     {
      Print("OpenCL not found");
      return;
     }
//--- Visualizamos la información sobre la plataforma
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_NAME,str))
      Print("OpenCL platform name: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_VENDOR,str))
      Print("OpenCL platform vendor: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_VERSION,str))
      Print("OpenCL platform ver: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_PROFILE,str))
      Print("OpenCL platform profile: ",str);
   if(CLGetInfoString(cl_ctx,CL_PLATFORM_EXTENSIONS,str))
      Print("OpenCL platform ext: ",str);
//--- Visualizamos la información sobre el dispositivo
   if(CLGetInfoString(cl_ctx,CL_DEVICE_NAME,str))
      Print("OpenCL device name: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_PROFILE,str))
      Print("OpenCL device profile: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_BUILT_IN_KERNELS,str))
      Print("OpenCL device kernels: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_EXTENSIONS,str))
      Print("OpenCL device ext: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_VENDOR,str))
      Print("OpenCL device vendor: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_VERSION,str))
      Print("OpenCL device ver: ",str);
   if(CLGetInfoString(cl_ctx,CL_DEVICE_OPENCL_C_VERSION,str))
      Print("OpenCL open c ver: ",str);
//--- Visualizamos la información general sobre el dispositivo OpenCL
   Print("OpenCL type: ",EnumToString((ENUM_CL_DEVICE_TYPE)CLGetInfoInteger(cl_ctx,CL
   Print("OpenCL vendor ID: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_VENDOR_ID));
   Print("OpenCL units: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_COMPUTE_UNITS));
   Print("OpenCL freq: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_MAX_CLOCK_FREQUENCY));
   Print("OpenCL global mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_GLOBAL_MEM_SIZE));
   Print("OpenCL local mem: ",CLGetInfoInteger(cl_ctx,CL_DEVICE_LOCAL_MEM_SIZE));
//---
  }
```

CLContextCreate

Crea el contexto OpenCL y devuelve el manejador para él.

```
int  CLContextCreate(
   int  device     // número ordinal del dispositivo OpenCL o macro
   );
```

Parámetros

device

[in] Número del dispositivo OpenCL en el sistema según el orden. En vez de un número concreto se puede indicar uno de los valores: CL_USE_ANY – se permite utilizar cualquier dispositivo disponible con el soporte de OpenCL; CL_USE_GPU_ONLY – la emulación OpenCL está prohibida y se permite utilizar sólo los dispositivos especializados con el soporte de OpenCL (tarjetas gráficas).

Valor devuelto

Manejador para el contexto OpenCL en caso del éxito, o -1 en caso del error. Para obtener la información sobre el error, utilice la función GetLastError().

CLContextFree

Elimina el contexto OpenCL.

```
void  CLContextFree(
   int  context     // manejador para el contexto OpenCL
   );
```

Parámetros

context

[in] Manejador del contexto OpenCL.

Valor devuelto

Ninguno. En caso de un error interno, se cambia el valor de _LastError. Para obtener la información sobre el error, llame a la función GetLastError().

CLProgramCreate

Crea un programa OpenCL desde el código fuente.

```
int  CLProgramCreate(
   int           context,     // manejador para el contexto OpenCL
   const string  source       // código fuente
   );
```

Parámetros

context

[in] Manejador del contexto OpenCL.

source

[in] Cadena con el código fuente OpenCL del programa.

Valor devuelto

Manejador para el objeto OpenCL en caso de ejecutarse con éxito. En caso del error devuelve -1. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

En estos momentos están previstos los siguientes códigos de errores:

- ERR_OPENCL_INVALID_HANDLE - un manejador inválido para context OpenCL,
- ERR_INVALID_PARAMETER - un parámetro literal inválido,
- ERR_NOT_ENOUGH_MEMORY - memoria insuficiente para completar la operación,
- ERR_OPENCL_PROGRAM_CREATE - un error interno de OpenCL o un error de compilación.

CLProgramFree

Elimina un programa OpenCL.

```
void  CLProgramFree(
   int  program     // manejador para el objeto OpenCL
   );
```

Parámetros

program

[in] Manejador del objeto OpenCL.

Valor devuelto

Ninguno. En caso de un error interno, se cambia el valor de _LastError. Para obtener la información sobre el error, llame a la función GetLastError().

CLKernelCreate

Crea un punto de acceso al programa OpenCL y devuelve el manejador para él.

```
int  CLKernelCreate(
   int           program,        // manejador para el objeto OpenCL
   const string  kernel_name     // nombre del kernel
   );
```

Parámetros

program

[in] Manejador para un objeto del programa OpenCL.

kernel_name

[in] Nombre de la función kernel, es decir el nombre del punto de acceso en el programa OpenCL correspondiente a partir del cual se empieza la ejecución.

Valor devuelto

Manejador para el objeto OpenCL en caso de ejecutarse con éxito. En caso del error devuelve -1. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

En estos momentos están previstos los siguientes códigos de errores:

- ERR_OPENCL_INVALID_HANDLE - manejador inválido para *program* OpenCL,
- ERR_INVALID_PARAMETER - un parámetro literal inválido,
- ERR_OPENCL_TOO_LONG_KERNEL_NAME - nombre del kernel contiene más de 127 caracteres,
- ERR_OPENCL_KERNEL_CREATE - un error interno a la hora de crear el objeto OpenCL.

CLKernelFree

Elimina la función del arranque OpenCL.

```
void  CLKernelFree(
   int  kernel      // manejador para el kernel del programa OpenCL
   );
```

Parámetros

kernel_name

[in] Manejador del objeto kernel.

Valor devuelto

Ninguno. En caso de un error interno, se cambia el valor de _LastError. Para obtener la información sobre el error, llame a la función GetLastError().

CLSetKernelArg

Establece un parámetro para la función OpenCL.

```
bool  CLSetKernelArg(
   int   kernel,        // manejador para el kernel del programa OpenCL
   uint  arg_index,     // número del argumento de la función OpenCL
   void  arg_value      // código fuente
   );
```

Parámetros

kernel

[in] Manejador para el kernel del programa OpenCL.

arg_index

[in] El número del argumento de la función, la numeración se comienza desde cero.

arg_value

[in] Valor del argumento de la función.

Valor devuelto

En caso del éxito la función devuelve true, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

En estos momentos están previstos los siguientes códigos de errores:

- ERR_INVALID_PARAMETER,
- ERR_OPENCL_INVALID_HANDLE – manejador inválido para el kernel OpenCL,
- ERR_OPENCL_SET_KERNEL_PARAMETER - error interno de OpenCL.

CLSetKernelArgMem

Establece el búfer OpenCL como parámetro de la función OpenCL.

```
bool  CLSetKernelArgMem(
   int   kernel,           // manejador para el kernel del programa OpenCL
   uint  arg_index,        // número del argumento de la función OpenCL
   int   cl_mem_handle     // manejador del búfer OpenCL
   );
```

Parámetros

kernel

[in] Manejador para el kernel del programa OpenCL.

arg_index

[in] El número del argumento de la función, la numeración se comienza desde cero.

cl_mem_handle

[in] Manejador para el búfer OpenCL.

Valor devuelto

En caso del éxito la función devuelve true, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

CLBufferCreate

Crea un búfer OpenCL y devuelve su manejador.

```
int  CLBufferCreate(
   int   context,     // manejador para el contexto OpenCL
   uint  size,        // tamaño del búfer
   uint  flags        // combinación de banderas que establecen las propiedades del b
   );
```

Parámetros

context

[in] Manejador para context OpenCL.

size

[in] Tamaño del búfer en bytes.

flags

[in] Las propiedades del búfer que se establecen mediante la combinación de banderas: CL_MEM_READ_WRITE, CL_MEM_WRITE_ONLY, CL_MEM_READ_ONLY, CL_MEM_ALLOC_HOST_PTR.

Valor devuelto

Manejador para el búfer OpenCL en caso de ejecutarse con éxito. En caso del error devuelve -1. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

En estos momentos están previstos los siguientes códigos de errores:

- ERR_OPENCL_INVALID_HANDLE - un manejador inválido para el contexto OpenCL,
- ERR_NOT_ENOUGH_MEMORY - memoria insuficiente,
- ERR_OPENCL_BUFFER_CREATE - un error interno de creación del búfer.

CLBufferFree

Elimina el búfer OpenCL.

```
void  CLBufferFree(
   int   buffer     // manejador para el búfer OpenCL
   );
```

Parámetros

buffer

[in] Manejador para el búfer OpenCL.

Valor devuelto

Ninguno. En caso de un error interno, se cambia el valor de _LastError. Para obtener la información sobre el error, llame a la función GetLastError().

CLBufferWrite

Escribe un array en el búfer OpenCL y devuelve el número de elementos escritos.

```
uint  CLBufferWrite(
   int           buffer,                  // manejador para el búfer OpenCL
   const void&   data[],                  // array de valores
   uint          buffer_offset=0,         // offset en el búfer OpenCL en bytes, por
   uint          data_offset=0,           // offset dentro del array en elementos, p
   uint          data_count=WHOLE_ARRAY   // número de valores del array para la esc
   );
```

Parámetros

buffer

[in] Manejador del búfer OpenCL.

data[]

[in] Array de valores que hay que escribir en el búfer OpenCL. Se pasa por referencia.

buffer_offset

[in] Desplazamiento (offset) en el búfer OpenCL en bytes, a partir del cual se empieza la escritura. Por defecto, la escritura se hace desde el principio del búfer.

data_offset

[in] Índice del primer elemento del array a partir del cual se cogen los valores desde el array para la escritura en el búfer OpenCL. Por defecto, los valores se cogen desde el principio del array.

data_count

[in] Número de valores que hay que escribir. Por defecto, todos los valores del array.

Valor devuelto

Número de valores que se han escrito. En caso del error, devuelve 0. Para obtener la información sobre el error, utilice la función GetLastError().

Nota

Para los arrays unidimensionales, el número del elemento a partir del que se empieza la lectura de dato para su escritura en el búfer OpenCL, se calcula teniendo en cuenta la bandera AS_SERIES.

Un array con dos o más dimensiones se representa como unidimensional. En este caso *data_offset* es el número de elementos que hay que saltar en la representación, y no el número de elementos en la primera dimensión.

CLBufferRead

Lee el búfer OpenCL en un array y devuelve el número de elementos leídos.

```
uint  CLBufferRead(
   int           buffer,                  // manejador para el búfer OpenCL
   const void&   data[],                  // array de valores
   uint          buffer_offset=0,         // offset en el búfer OpenCL en bytes, por
   uint          data_offset=0,           // offset dentro del array en elementos, p
   uint          data_count=WHOLE_ARRAY   // número de valores del búfer para la lec
   );
```

Parámetros

buffer

[in] Manejador del búfer OpenCL.

data[]

[in] Array para recibir valores desde el búfer OpenCL. Se pasa por referencia.

buffer_offset

[in] Desplazamiento (offset) en el búfer OpenCL en bytes, a partir del cual se empieza la lectura. Por defecto, la lectura se empieza desde el principio del búfer.

data_offset

[in] Índice del primer elemento del array para la escritura de los valores del búfer OpenCL. Por defecto, los valores leídos se escriben en el array desde el índice cero.

data_count

[in] Número de valores que hay que leer. Por defecto, se lee el búfer OpenCL entero.

Valor devuelto

Número de valores leídos. En caso del error, devuelve 0. Para obtener la información sobre el error, utilice la función GetLastError().

Nota

Para los arrays unidimensionales, el número del elemento en el que se empieza la escritura de datos desde el búfer OpenCL, se calcula teniendo en cuenta la bandera AS_SERIES.

Un array con dos o más dimensiones se representa como unidimensional. En este caso *data_offset* es el número de elementos que hay que saltar en la representación, y no el número de elementos en la primera dimensión.

CLExecute

Ejecuta un programa OpenCL. Hay 3 variantes de esta función:

1. Inicio de la función kernel usando un solo núcleo

```
bool  CLExecute(
   int          kernel                  // manejador para el kernel de un programa (
   );
```

2. Inicio de varias copias kernel (función OpenCL) con la descripción del espacio de tareas

```
bool  CLExecute(
   int          kernel,                 // manejador para el kernel del programa Op
   uint         work_dim,               // dimensión del espacio de tareas
   const uint&  global_work_offset[],   // offset inicial en el espacio de tareas
   const uint&  global_work_size[]      // número total de tareas
   );
```

3. Inicio de varias copias kernel (función OpenCL) con la descripción del espacio de tareas y especificación del tamaño del subconjunto local de tareas en grupo

```
bool  CLExecute(
   int          kernel,                 // manejador para el kernel del programa Op
   uint         work_dim,               // dimensión del espacio de tareas
   const uint&  global_work_offset[],   // offset inicial en el espacio de tareas
   const uint&  global_work_size[],     // número total de tareas
   const uint&  local_work_size[]       // número de tareas en el grupo local
   );
```

Parámetros

kernel

[in] Manejador para el kernel OpenCL.

work_dim

[in] Dimensión del espacio de tareas.

global_work_offset[]

[in] Desplazamiento inicial en el espacio de tareas.

global_work_size[]

[in] Tamaño del subconjunto de tareas.

local_work_size[]

[in] Tamaño del subconjunto local de tareas en el grupo.

Valor devuelto

En caso del éxito la función devuelve true, de lo contrario devuelve false. Para obtener la información sobre el error, hay que llamar a la función GetLastError().

Nota

Vamos a ver el uso de los parámetros con la ayuda de los siguientes ejemplos:

- *work_dim* establece la dimensión del array *work_items[]* que describe las tareas. Si *work_dim*=3, se usa el array de tres dimensiones *work_items[N1, N2, N3]*.
- *global_work_size[]* contiene los valores que establecen el tamaño del array *work_items[]*. Si tenemos work_dim=3, y en consecuencia, el array *global_work_size[3]* puede ser {40, 100, 320}. Entonces tenemos *work_items[40, 100, 320]*. Eso significa que el número total de tareas es igual a 40 x 100 x 320 = 1 280 000.
- *local_work_size[]* establece el conjunto de tareas que van a ejecutarse por el kernel especificado del programa OpenCL. Su dimensión es igual a la dimensión *work_items[]* y permite dividir el subconjunto general de las tareas en los subconjuntos más pequeños sin restas de la división. Prácticamente, los tamaños del array *local_work_size[]* tiene que ser seleccionado de tal manera que el conjunto global de tareas *work_items[]* se divida en los subconjuntos del menor tamaño. En este ejemplo queda bien *local_work_size[3]={10, 10, 10}*, ya que *work_items[40, 100, 320]* se puede reunir sin la resta desde el array *local_items[10, 10, 10]*.

Standard Library

This group of chapters contains the technical details of the MQL5 Standard Library and descriptions of all its key components.

MQL5 Standard Library is written in MQL5 and is designed to facilitate writing programs (indicators, scripts, experts) to end users. Library provides convenient access to most of the internal functions MQL5.

MQL5 Standard Library is placed in the working directory of the terminal in the 'Include' folder.

Section	Location
Base class	Include\
Classes of data	Include\Arrays\
Classes for file operations	Include\Files\
Classes for string operations	Include\Strings\
Classes for graphic objects	Include\Objects\
Класс для создания пользовательской графики	Include\Canvas\
Class for working with chart	Include\Charts\
Technical indicators	Include\Indicators\
Trade classes	Include\Trade\
Trading strategy classes	Include\Expert\
Classes for creation of control panels and dialogs	Include\Controls\

Basic Class CObject

Class CObject is the base class for constructing a MQL5 Standard Library .

Description

Class CObject provides all its descendants to be part of a linked list. Also identifies a number of virtual methods for further implementation in descendant classes.

Declaration

```
class CObject
```

Title

```
#include <Object.mqh>
```

Class Methods

Attributes	
Prev	Gets the value of the previous item
Prev	Sets the value of the previous item
Next	Gets the value of the subsequent element
Next	Sets the next element
Compare methods	
virtual Compare	Returns the result of comparison with another object
Input/output	
virtual Save	Writes object to a file
virtual Load	Reads the object from the file
virtual Type	Returns the type of object

Derived classes:

- CArray
- CChartObject
- CChart
- CString
- CFile
- CList
- CTreeNode

Prev

Gets a pointer to the previous list item.

```
CObject*  Prev()
```

Return Value

Pointer to the previous list item. If an item is listed first, then return NULL.

Example:

```
//--- example for CObject::Prev()
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- use prev object
   CObject *object=object_second.Prev();
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Prev

Sets the pointer to the previous list item.

```
void  Prev(
   CObject*  object      // Pointer to the previous list item
   )
```

Parameters

object

[in] New value pointer to the previous list item.

Example:

```
//--- example for CObject::Prev(CObject*)
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- use objects
   //--- ...
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Next

Gets a pointer to the next element of the list.

```
CObject*  Next()
```

Return Value

Pointer to the next item in the list. If the last item in the list, returns NULL.

Example:

```
//--- example for CObject::Next()
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- use next object
   CObject *object=object_first.Next();
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Next

Sets the pointer to the next element of the list.

```
void  Next(
   CObject*  object      // Pointer to the next list item
   )
```

Parameters

object

[in] New value pointer to the next list item.

Example:

```
//--- example for CObject::Next(CObject*)
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
   object_second.Prev(object_first);
   //--- use objects
   //--- ...
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Compare

Compares the data item in the list with data on another element of the list.

```
virtual int  Compare(
   CObject const *  node,      // Node to compare with
   int              mode=0     // Compare mode
   ) const
```

Parameters

node

[in] Pointer to a list item to compare

mode=0

[in] Variant Comparison

Return Value

0 - in case the list items are equal, -1 - if the list item is less than the item in the list for comparison (node), 1 - if the list item more than item in the list for comparison (node).

Note

Method Compare () in class CObject always returns 0 and does not perform any action.If you want to compare data derived class, the method Compare (...) Should be implemented.The mode parameter should be used when implementing multivariate comparison.

Example:

```
//--- example for CObject::Compare(...)
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object_first,*object_second;
   //---
   object_first=new CObject;
   if(object_first==NULL)
     {
      printf("Object create error");
      return;
     }
   object_second=new CObject;
   if(object_second==NULL)
     {
      printf("Object create error");
      delete object_first;
      return;
     }
   //--- set interconnect
   object_first.Next(object_second);
```

```
   object_second.Prev(object_first);
   //--- compare objects
   int result=object_first.Compare(object_second);
   //--- delete objects
   delete object_first;
   delete object_second;
  }
```

Save

Saves data element list in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (), binary

Return Value

true - if successfully completed, false - if an error.

Note

Method Save (int) in class CObject always returns true and does not perform any action. If you want to save the data derived class in the file, the method Save (int) should be implemented.

Example:

```
//--- example for CObject::Save(int)
#include <Object.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CObject *object=new CObject;
   //---
   if(object!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- set objects data
   //--- . . .
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete object;
         FileClose(file_handle);
         //---
         return;
        }
```

```
      FileClose(file_handle);
     }
   delete object;
  }
```

Load

Loads data item in the list from a file.

```
virtual bool  Load(
   int  file_handle      // handle to file
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (), binary

Return Value

true - if successfully completed, false - if an error.

Note

Method Load (int) in class CObject always returns true and does not perform any action. If you want to load the data derived class from a file, the method Load (int) should be implemented.

Example:

```
//--- example for CObject::Load(int)
#include <Object.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CObject *object=new CObject;
   //---
   if(object!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete object;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
```

```
   //--- use object
   //--- . . .
   delete object;
  }
```

Type

Gets the type identifier.

```
virtual int  Type() const
```

Return Value

Type identifier (for CObject - 0).

Example:

```
//--- example for CObject::Type()
#include <Object.mqh>
//---
void OnStart()
  {
   CObject *object=new CObject;
   //---
   object=new CObject;
   if(object ==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get objects type
   int type=object.Type();
   //--- delete object
   delete object;
  }
```

Data Structures

This section contains the technical details of working with various data structures (arrays, linked lists, etc.) and description of the relevant components of the MQL5 Standard Library .

Using classes of data structures, will save time when creating custom data stores of various formats (including composite data structures).

MQL5 Standard Library (in terms of data sets) is placed in the working directory of the terminal in the Include\Arrays folder.

Data Arrays

Use of classes of dynamic arrays of data will save time when creating a custom data stores of various formats (including multidimensional arrays).

MQL5 Standard Library (in terms of arrays of data) is placed in the working directory of the terminal in the Include\Arrays folder.

Class	Description
Base class of dynamic data array CArray	Base class of dynamic data array
CArrayChar	Dynamic array of variables of type char or uchar
CArrayShort	Dynamic array of variables of type short or ushort
CArrayInt	Dynamic array of variables of type int or uint
CArrayLong	Dynamic array of variables of type long or ulong
CArrayFloat	Dynamic array of variables of type float
CArrayDouble	Dynamic array of variables of type double
CArrayString	Dynamic array of variables of type string
Base class of object array CArrayObj	Dynamic array of pointers CObject
Base class of list CList	Provides the ability to work with a list of instances of CObject and its descendant
CTreeNode	Provides the ability to work with nodes of the binary tree CTree
CTree	Provides the ability to work with the binary tree of the CTreeNode class instances and its descendants

CArray

CArray class is the base class a dynamic array of variables.

Description

Class CArray provides the ability to work with a dynamic array of variables in the part of the attributes of management of memory allocation, sorting, and working with files.

Declaration

```
class CArray : public CObject
```

Title

```
#include <Arrays\Array.mqh>
```

Class Methods

Attributes	
Step	Gets the step increment size of the array
Step	Set the increment size of the array
Total	Gets the number of elements in the array
Available	Gets the number of free elements of the array are available without additional memory allocation
Max	Gets the maximum possible size of the array without memory reallocation
IsSorted	Gets sign sorted array to the specified option
SortMode	Gets the version of the sorting array
Clear methods	
Clear	Deletes all of the array elements without memory release
Sort methods	
Sort	Sorts an array to the specified option
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from a file

Derived classes:

- CArrayChar

- CArrayShort
- CArrayInt
- CArrayLong
- CArrayFloat
- CArrayDouble
- CArrayString
- CArrayObj

Step

Gets the step increment size of the array.

```
int  Step() const
```

Return Value

Increment size of the array.

Example:

```
//--- example for CArray::Step()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get resize step
   int step=array.Step();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Step

Sets the increment size of the array.

```
bool  Step(
   int  step      // step
   )
```

Parameters

step

[in] The new value of step increments in the size of the array.

Return Value

true if successful, false - if there was an attempt to establish a step less than or equal to zero.

Example:

```
//--- example for CArray::Step(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- set resize step
   bool result=array.Step(1024);
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Total

Gets the number of elements in the array.

```
int  Total() const;
```

Return Value

Number of elements in the array.

Example:

```
//--- example for CArray::Total()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check total
   int total=array.Total();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Available

Gets the number of free elements of the array are available without additional memory allocation.

```
int  Available() const
```

Return Value

Number of free elements of the array are available without additional memory allocation.

Example:

```
//--- example for CArray::Available()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check available
   int available=array.Available();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Max

Gets the maximum possible size of the array without memory reallocation.

```
int  Max() const
```

Return Value

The maximum possible size of the array without reallocation memory.

Example:

```
//--- example for CArray::Max()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check maximum size
   int max=array.Max();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

IsSorted

Gets sign sorted array to the specified option.

```
bool  IsSorted(
   int  mode=0      // Sorting mode
   ) const
```

Parameters

mode=0

[in] Tested version of the sort.

Return Value

Flag of the sorted list. If the list is sorted by the specified version - true, otherwise - false.

Note

Symptom sorted array can not be changed directly. Symptom sorted set method Sort () and reset any methods to add / insert except InsertSort (...).

Example:

```
//--- example for CArray::IsSorted()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check sorted
   if(array.IsSorted())
     {
      //--- use methods for sorted array
      //--- ...
     }
   //--- delete array
   delete array;
  }
```

SortMode

Gets the version of the sorting array.

```
int  SortMode() const;
```

Return Value

Sorting mode.

Example:

```
//--- example for CArray::SortMode()
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check sort mode
   int sort_mode=array.SortMode();
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Clear

Deletes all of the array elements without memory release.

```
void  Clear()
```

Return Value

None.

Example:

```
//--- example for CArray::Clear()
#include <Arrays\Array.mqh>
//---
void OnStart()
 {
 CArray *array=new CArray;
 //---
 if(array==NULL)
 {
 printf("Object create error");
 return;
 }
 //--- use array
 //--- ...
 //--- clear array
 array.Clear();
 //--- delete array
 delete array;
 }
```

Sort

Sorts an array to the specified option.

```
void  Sort(
   int  mode=0      // Sorting mode
   )
```

Parameters

mode=0

[in] Mode of array sorting.

Return Value

No.

Note

Sorting an array is always ascending.For arrays of primitive data types (CArrayChar, CArrayShort, etc.), the parameter mode is not used.For the array CArrayObj, multivariate sort should be implemented in the method Sort (int) derived class.

Example:

```
//--- example for CArray::Sort(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   CArray *array=new CArray;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- sorting by mode 0
   array.Sort(0);
   //--- use array
   //--- ...
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArray::Save(int)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   int     file_handle;
   CArray *array=new CArray;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- delete array
   delete array;
  }
```

Load

Loads data array from a file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArray::Load(...)
#include <Arrays\Array.mqh>
//---
void OnStart()
  {
   int     file_handle;
   CArray *array=new CArray;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- delete array
   delete array;
  }
```

CArrayChar

CArrayChar class is a class of dynamic array of variables of type char or uchar.

Description

Class CArrayChar provides the ability to work with a dynamic array of variables of type char or uchar.In the class implemented the ability to add / insert / delete elements in an array, sort array, searching in sorted array.In addition, the implemented methods of work with the file.

Declaration

```
class CArrayChar : public CArray
```

Title

```
#include <Arrays\ArrayChar.mqh>
```

Class Methods

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds to the end of the array elements from another array
AddArray	Adds to the end of the array elements from another array
Insert	Inserts an element in the array to the specified position
InsertArray	Inserts an array of elements from another array with the specified position
InsertArray	Inserts an array of elements from another array with the specified position
AssignArray	Copies the array elements from another array
AssignArray	Copies the array elements from another array
Modify methods	
Update	Changes the element at the specified position array

Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified position array
DeleteRange	Deletes a group of elements from the specified position array
Access methods	
At	Gets the element from the specified position array
Compare methods	
CompareArray	Compares array with another array
CompareArray	Compares array with another array
Sorted array methods	
InsertSort	Inserts element in a sorted array
Search	Searches for an element equal to the sample in the sorted array
SearchGreat	Searches for an element of more samples in sorted array
SearchLess	Searches for an element less than the sample in the sorted array
SearchGreatOrEqual	Searches for an element greater than or equal to the model in sorted array
SearchLessOrEqual	Searches for an element less than or equal to the model in sorted array
SearchFirst	Finds the first element equal to the model in sorted array
SearchLast	Finds the last element equal to the model in sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from a file
virtual Type	Gets the type identifier of the array

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // Number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true if successful, false - if there was an attempt to seek the amount is less than or equal to zero, or if the array did not increase.

Note

To reduce memory fragmentation, increase the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayChar::Reserve(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // Size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true if successful, false - if there was an attempt to set the size of less than or equal to zero.

Note

Changing the size of the array allows optimal use of memory.Superfluous elements on the right lost.To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayChar::Resize(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true if successful, false - if an error occurred.

Example:

```
//--- example for CArrayChar::Shutdown()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   char  element      // Element to add
   )
```

Parameters

element

[in] value of the element to add to the array.

Return Value

true if successful, false - if you can not add an element.

Example:

```
//--- example for CArrayChar::Add(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const char&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayChar::AddArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const CArrayChar*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayChar-source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayChar::AddArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayChar *src=new CArrayChar;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
```

```
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element in the array to the specified position.

```
bool  Insert(
   char  element,     // Element to insert
   int   pos          // Position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayChar::Insert(char,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array with the specified position.

```
bool  InsertArray(
   const char&  src[],     // Source array
   int           pos       // Position
   )
```

Parameters

src[]

[in] Reference to an array of source elements to insert

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayChar::InsertArray(const char &[],int)
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array with the specified position.

```
bool  InsertArray(
   CArrayChar*  src,      // Pointer to the source
   int           pos      // Position
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayChar-source elements to insert.

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayChar::InsertArray(const CArrayChar*,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayChar *src=new CArrayChar;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
```

```
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const char&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayChar::AssignArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const CArrayChar*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayChar-source element to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayChar::AssignArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayChar *src  =new CArrayChar;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
```

```
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified position array.

```
bool  Update(
   int   pos,          // Position
   char  element       // Value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value element

Return Value

true if successful, false - if you can not change the element.

Example:

```
//--- example for CArrayChar::Update(int,char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,'A'))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // Position
   int  shift       // Value
   )
```

Parameters

pos

[in] Position of the moved element array

shift

[in] The value of displacement (both positive and negative).

Return Value

true if successful, false - if you can not move the item.

Example:

```
//--- example for CArrayChar::Shift(int,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the given position in the array.

```
bool  Delete(
   int  pos      // Position
   )
```

Parameters

pos

[in] Position removes the element in the array.

Return Value

true if successful, false - if you can not remove the element.

Example:

```
//--- example for CArrayChar::Delete(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from a given position in the array.

```
bool  DeleteRange(
   int  from,     // Position of the first element
   int  to        // Positions of the last element
   )
```

Parameters

from

[in] Position of the first removes the element in the array.

to

[in] Position of the last deleted element in the array.

Return Value

true if successful, false - if you can not remove elements.

Example:

```
//--- example for CArrayChar::DeleteRange(int,int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the given position in the array.

```
char  At(
   int  pos      // Position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element in case of success, CHAR_MAX-if there was an attempt to get an element of not existing positions (the last error ERR_OUT_OF_RANGE).

Note

Of course, CHAR_MAX may be a valid value of an array element, so having a value, always check the last error code.

Example:

```
//--- example for CArrayChar::At(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      char result=array.At(i);
      if(result==CHAR_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- error of reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
```

```
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const char&  src[]      // Source array
   ) const
```

Parameters

src[]

[in] Reference to an array of source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayChar::CompareArray(const char &[])
#include <Arrays\ArrayChar.mqh>
//---
char src[];
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const CArrayChar*  src      // Pointer to the sources
   ) const
```

Parameters

src

[in] Pointer to an instance of class CArrayChar-source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayChar::CompareArray(const CArrayChar*)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayChar *src=new CArrayChar;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts element in a sorted array.

```
bool  InsertSort(
   char  element      // Element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayChar::InsertSort(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort('A'))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   char  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayChar::Search(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search('A')!=-1) printf("Element found");
   else                      printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element of more samples in sorted array.

```
int  SearchGreat(
   char  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayChar::SearchGreat(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat('A')!=-1) printf("Element found");
   else                           printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element less than the sample in the sorted array.

```
int  SearchLess(
   char  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayChar::SearchLess(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess('A')!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element greater than or equal to the sample in the sorted array

```
int  SearchGreatOrEqual(
   char  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayChar::SearchGreatOrEqual(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual('A')!=-1) printf("Element found");
   else                                  printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element less than or equal to the sample in the sorted array.

```
int  SearchLessOrEqual(
   char  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayChar::SearchLessOrEqual(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual('A')!=-1) printf("Element found");
   else                                 printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Finds the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   char  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayChar::SearchFirst(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst('A')!=-1) printf("Element found");
   else                           printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Finds the last element equal to the model in sorted array.

```
int  SearchLast(
   char  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayChar::SearchLast(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast('A')!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   char  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the element was not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayChar::SearchLinear(char)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear('A')!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayChar::Save(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayChar *array=new CArrayChar;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- delete array
   delete array;
  }
```

Load

Loads data array from a file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayChar::Load(int)
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayChar *array=new CArrayChar;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = '%c'",i,array.At(i));
```

```
      }
   //--- delete array
   delete array;
  }
```

Type

Gets the type identifier of the array.

```
virtual int  Type() const
```

Return Value

ID type of the array (for CArrayChar - 77).

Example:

```
//--- example for CArrayChar::Type()
#include <Arrays\ArrayChar.mqh>
//---
void OnStart()
  {
   CArrayChar *array=new CArrayChar;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayShort

CArrayShort class is a class of dynamic array of variables of type short or ushort.

Description

Class CArrayShort provides the ability to work with a dynamic array of variables of type short or ushort.In the class implemented the ability to add / insert / delete elements in an array, sort array, searching in sorted array.In addition, the implemented methods of work with the file.

Declaration

```
class CArrayShort : public CArray
```

Title

```
#include <Arrays\ArrayShort.mqh>
```

Class Methods

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds to the end of the array elements from another array
AddArray	Adds to the end of the array elements from another array
Insert	Inserts an element in the array to the specified position
InsertArray	Inserts an array of elements from another array with the specified position
InsertArray	Inserts an array of elements from another array with the specified position
AssignArray	Copies the array elements from another array
AssignArray	Copies the array elements from another array
Update methods	
Update	Changes the element at the specified position array

Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified position array
DeleteRange	Deletes a group of elements from the specified position array
Access methods	
At	Gets the element from the specified position array
Compare methods	
CompareArray	Compares array with another array
CompareArray	Compares array with another array
Sorted array operations	
InsertSort	Inserts element in a sorted array
Search	Searches for an element equal to the model in sorted array
SearchGreat	Searches for an element of more samples in sorted array
SearchLess	Searches for an element less than the sample in the sorted array
SearchGreatOrEqual	Searches for an element greater than or equal to the model in sorted array
SearchLessOrEqual	Searches for an element less than or equal to the model in sorted array
SearchFirst	Finds the first element equal to the model in sorted array
SearchLast	Finds the last element equal to the model in sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from a file
virtual Type	Gets the type identifier of the array

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // Number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true if successful, false - if there was an attempt to seek the amount is less than or equal to zero, or if the array did not increase.

Note

To reduce memory fragmentation, increase the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayShort::Reserve(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // Size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true if successful, false - if there was an attempt to set the size of less than or equal to zero.

Note

Changing the size of the array allows optimal use of memory.Superfluous elements on the right lost.To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayShort::Resize(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true if successful, false - if an error occurred.

Example:

```
//--- example for CArrayShort::Shutdown()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   short  element      // Element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true if successful, false - if you can not add an element.

Example:

```
//--- example for CArrayShort::Add(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const short&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayShort::AddArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const CArrayShort*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayShort-source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayShort::AddArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayShort *src=new CArrayShort;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
```

```
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element in the array to the specified position.

```
bool  Insert(
   short  element,     // Element to insert
   int    pos          // Position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayShort::Insert(short,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array specified position.

```
bool  InsertArray(
   const short&  src[],     // Source array
   int           pos        // Position
   )
```

Parameters

src[]

[in] Reference to an array of source elements to insert

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayShort::InsertArray(const short &[],int)
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array ukazannogy position.

```
bool  InsertArray(
   CArrayShort*  src,      // Pointer to the source
   int            pos      // Position
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayShort-source elements to insert.

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayShort::InsertArray(const CArrayShort*,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayShort *src=new CArrayShort;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
```

```
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const short&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayShort::AssignArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const CArrayShort*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayShort-source element to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayShort::AssignArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayShort *src  =new CArrayShort;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
```

```
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified position array.

```
bool  Update(
   int    pos,          // Position
   short  element       // Value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value element

Return Value

true if successful, false - if you can not change the element.

Example:

```
//--- example for CArrayShort::Update(int,short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,100))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // Positions
   int  shift       // Shift
   )
```

Parameters

pos

[in] Position of the moved element array

shift

[in] The value of displacement (both positive and negative).

Return Value

true if successful, false - if you can not move the item.

Example:

```
//--- example for CArrayShort::Shift(int,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the given position in the array.

```
bool  Delete(
   int  pos      // Position
   )
```

Parameters

pos

[in] Position removes the element in the array.

Return Value

true if successful, false - if you can not remove the element.

Example:

```
//--- example for CArrayShort::Delete(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from a given position in the array.

```
bool  DeleteRange(
   int  from,     // Position of the first element
   int  to        // Position of the lase element
   )
```

Parameters

from

[in] Position of the first removes the element in the array.

to

[in] Position of the last deleted element in the array.

Return Value

true if successful, false - if you can not remove elements.

Example:

```
//--- example for CArrayShort::DeleteRange(int,int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the given position in the array.

```
short  At(
   int  pos      // Position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element in case of success, SHORT_MAX-if there was an attempt to get an element of not existing positions (the last error ERR_OUT_OF_RANGE).

Note

Of course, SHORT_MAX may be a valid value of an array element, so having a value, always check the last error code.

Example:

```
//--- example for CArrayShort::At(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      short result=array.At(i);
      if(result==SHORT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- error of reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
```

```
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const short&  src[]      // Source array
   ) const
```

Parameters

src[]

[in] Reference to an array of source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayShort::CompareArray(const short &[])
#include <Arrays\ArrayShort.mqh>
//---
short src[];
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const CArrayShort*  src      // Pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of class CArrayShort-source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayShort::CompareArray(const CArrayShort*)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayShort *src=new CArrayShort;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts element in a sorted array.

```
bool  InsertSort(
   short  element      // Element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayShort::InsertSort(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(100))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   short  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayShort::Search(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(100)!=-1) printf("Element found");
   else                      printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element of more samples in sorted array.

```
int  SearchGreat(
   short  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayShort::SearchGreat(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(100)!=-1) printf("Element found");
   else                           printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element less than the sample in the sorted array.

```
int  SearchLess(
   short  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayShort::SearchLess(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(100)!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element greater than or equal to the sample in the sorted array.

```
int  SearchGreatOrEqual(
   short  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayShort::SearchGreatOrEqual(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(100)!=-1) printf("Element found");
   else                                  printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element less than or equal to the sample in the sorted array.

```
int  SearchLessOrEqual(
   short  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayShort::SearchLessOrEqual(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(100)!=-1) printf("Element found");
   else                                 printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Finds the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   short  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayShort::SearchFirst(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(100)!=-1) printf("Element found");
   else                           printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Finds the last element equal to the model in sorted array.

```
int  SearchLast(
   short  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayShort::SearchLast(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(100)!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   short  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the element was not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayShort::SearchLinear(short)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(100)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayShort::Save(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayShort *array=new CArrayShort;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

```
      }
   delete array;
  }
```

Load

Loads data array from a file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayShort::Load(int)
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayShort *array=new CArrayShort;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %d",i,array.At(i));
```

```
      }
   delete array;
  }
```

Type

Gets the type identifier of the array.

```
virtual int  Type() const
```

Return Value

ID type of the array (for CArrayShort - 82).

Example:

```
//--- example for CArrayShort::Type()
#include <Arrays\ArrayShort.mqh>
//---
void OnStart()
  {
   CArrayShort *array=new CArrayShort;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayInt

CArrayInt class is a class of dynamic array of variables of type int or uint.

Description

Class CArrayInt provides the ability to work with a dynamic array of variables of type int or uint.In the class implemented the ability to add / insert / delete elements in an array, sort array, searching in sorted array.In addition, the implemented methods of work with the file.

Declaration

```
class CArrayInt : public CArray
```

Title

```
#include <Arrays\ArrayInt.mqh>
```

Class Methods

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds to the end of the array elements from another array
AddArray	Adds to the end of the array elements from another array
Insert	Inserts an element in the array to the specified position
InsertArray	Inserts an array of elements from another array with the specified position
InsertArray	Inserts an array of elements from another array with the specified position
AssignArray	Copies the array elements from another array
AssignArray	Copies the array elements from another array
Update methods	
Update	Changes the element at the specified position array

Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified position array
DeleteRange	Deletes a group of elements from the specified position array
Access methods	
At	Gets the element from the specified position array
Compare methods	
CompareArray	Compares array with another array
CompareArray	Compares array with another array
Sorted array operations	
InsertSort	Inserts element in a sorted array
Search	Searches for an element equal to the model in sorted array
SearchGreat	Searches for an element of more samples in sorted array
SearchLess	Searches for an element less than the sample in the sorted array
SearchGreatOrEqual	Searches for an element greater than or equal to the model in sorted array
SearchLessOrEqual	Searches for an element less than or equal to the model in sorted array
SearchFirst	Finds the first element equal to the model in sorted array
SearchLast	Finds the last element equal to the model in sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from a file
virtual Type	Gets the type identifier of the array

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // Number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true if successful, false - if there was an attempt to seek the amount is less than or equal to zero, or if the array did not increase.

Note

To reduce memory fragmentation, increase the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayInt::Reserve(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // Number
   )
```

Parameters

size

[in] New size of the array.

Return Value

true if successful, false - if there was an attempt to set the size of less than or equal to zero.

Note

Changing the size of the array allows optimal use of memory.Superfluous elements on the right lost.To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayInt::Resize(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true if successful, false - if an error occurred.

Example:

```
//--- example for CArrayInt::Shutdown()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   int  element      // Element to add
   )
```

Parameters

element

[in] value of the element to add to the array.

Return Value

true if successful, false - if you can not add an element.

Example:

```
//--- example for CArrayInt::Add(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const int&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayInt::AddArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const CArrayInt*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayInt-source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayInt::AddArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayInt *src=new CArrayInt;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
```

```
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element in the array to the specified position.

```
bool  Insert(
   int  element,     // Element to insert
   int  pos          // Position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayInt::Insert(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array specified position.

```
bool  InsertArray(
   const int&  src[],     // Source array
   int          pos        // Position
   )
```

Parameters

src[]

[in] Reference to an array of source elements to insert

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayInt::InsertArray(const int &[],int)
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array specified position.

```
bool  InsertArray(
   CArrayInt*  src,     // Pointer to the source
   int          pos      // Position
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayInt-source elements to insert.

pos

[in] Position in the array to insert.

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayInt::InsertArray(const CArrayInt*,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayInt *src=new CArrayInt;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
```

```
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const int&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayInt::AssignArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const CArrayInt*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayInt-source element to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayInt::AssignArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayInt *src  =new CArrayInt;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
```

```
   //--- arrays is identical
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified position array.

```
bool  Update(
   int  pos,          // Position
   int  element       // Value
   )
```

Parameters

pos

[in] Position of the element in the array to change.

element

[in] New value element

Return Value

true if successful, false - if you can not change the element.

Example:

```
//--- example for CArrayInt::Update(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,10000))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // Position
   int  shift       // Shift
   )
```

Parameters

pos

[in] Position of the moved element array

shift

[in] The value of displacement (both positive and negative).

Return Value

true if successful, false - if you can not move the item.

Example:

```
//--- example for CArrayInt::Shift(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the given position in the array.

```
bool  Delete(
   int  pos      // Position
   )
```

Parameters

pos

[in] Position removes the element in the array.

Return Value

true if successful, false - if you can not remove the element.

Example:

```
//--- example for CArrayInt::Delete(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from a given position in the array.

```
bool  DeleteRange(
   int  from,     // Position of the first element
   int  to        // Position of the last element
   )
```

Parameters

from

[in] Position of the first removes the element in the array.

to

[in] Position of the last deleted element in the array.

Return Value

true if successful, false - if you can not remove elements.

Example:

```
//--- example for CArrayInt::DeleteRange(int,int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the given position in the array.

```
int  At(
   int  pos      // Position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element in case of success, INT_MAX-if there was an attempt to get an element of not existing positions (the last error ERR_OUT_OF_RANGE).

Note

Of course, INT_MAX could be a valid value of an array element, so having a value, always check the last error code.

Example:

```
//--- example for CArrayInt::At(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      int result=array.At(i);
      if(result==INT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- error of reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
```

```
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const int&  src[]      // Source array
   ) const
```

Parameters

src[]

[in] Reference to an array of source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayInt::CompareArray(const int &[])
#include <Arrays\ArrayInt.mqh>
//---
int src[];
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const CArrayInt*  src      // Pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of class CArrayInt-source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayInt::CompareArray(const CArrayInt*)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayInt *src=new CArrayInt;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts element in a sorted array.

```
bool  InsertSort(
   int  element      // Element to insert
   )
```

Parameters

element

[in] value of the element to be inserted into a sorted array

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayInt::InsertSort(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(10000))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   int  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayInt::Search(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(10000)!=-1) printf("Element found");
   else                        printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element of more samples in sorted array.

```
int  SearchGreat(
   int  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayInt::SearchGreat(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(10000)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element less than the sample in the sorted array.

```
int  SearchLess(
   int  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayInt::SearchLess(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(10000)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element greater than or equal to the sample in the sorted array.

```
int  SearchGreatOrEqual(
   int  element      // Element to search
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayInt::SearchGreatOrEqual(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(10000)!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element less than or equal to the sample in the sorted array.

```
int  SearchLessOrEqual(
   int  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayInt::SearchLessOrEqual(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(10000)!=-1) printf("Element found");
   else                                   printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Finds the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   int  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayInt:: SearchFirst(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(10000)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Finds the last element equal to the model in sorted array.

```
int  SearchLast(
   int  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayInt::SearchLast(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(10000)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   int  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the element was not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayInt::SearchLinear(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(10000)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayInt::Save(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   int        file_handle;
   CArrayInt *array=new CArrayInt;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

```
      }
   delete array;
  }
```

Load

Loads data array from a file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayInt::Load(int)
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   int        file_handle;
   CArrayInt *array=new CArrayInt;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %d",i,array.At(i));
```

```
      }
   delete array;
  }
```

Type

Gets the type identifier of the array.

```
virtual int  Type() const
```

Return Value

ID type of the array (for CArrayInt - 82).

Example:

```
//--- example for CArrayInt::Type()
#include <Arrays\ArrayInt.mqh>
//---
void OnStart()
  {
   CArrayInt *array=new CArrayInt;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayLong

CArrayLong class is a class of dynamic array of variables of type long or ulong.

Description

Class CArrayLong provides the ability to work with a dynamic array of variables of type long or ulong.In the class implemented the ability to add / insert / delete elements in an array, sort array, searching in sorted array.In addition, the implemented methods of work with the file.

Declaration

```
class CArrayLong : public CArray
```

Title

```
#include <Arrays\ArrayLong.mqh>
```

Class Methods

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds to the end of the array elements from another array
AddArray	Adds to the end of the array elements from another array
Insert	Inserts an element in the array to the specified position
InsertArray	Inserts an array of elements from another array with the specified position
InsertArray	Inserts an array of elements from another array with the specified position
AssignArray	Copies the array elements from another array
AssignArray	Copies the array elements from another array
Update methods	
Update	Changes the element at the specified position array

Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified position array
DeleteRange	Deletes a group of elements from the specified position array
Access methods	
At	Gets the element from the specified position array
Compare methods	
CompareArray	Compares array with another array
CompareArray	Compares array with another array
Sorted array operations	
InsertSort	Inserts element in a sorted array
Search	Searches for an element equal to the model in sorted array
SearchGreat	Searches for an element of more samples in sorted array
SearchLess	Searches for an element less than the sample in the sorted array
SearchGreatOrEqual	Searches for an element greater than or equal to the model in sorted array
SearchLessOrEqual	Searches for an element less than or equal to the model in sorted array
SearchFirst	Finds the first element equal to the model in sorted array
SearchLast	Finds the last element equal to the model in sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from a file
virtual Type	Gets the type identifier of the array

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // Number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true if successful, false - if there was an attempt to seek the amount is less than or equal to zero, or if the array did not increase.

Note

To reduce memory fragmentation, increase the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayLong::Reserve(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // Size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true if successful, false - if there was an attempt to set the size of less than or equal to zero.

Note

Changing the size of the array allows optimal use of memory.Superfluous elements on the right lost.To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayLong::Resize(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true if successful, false - if an error occurred.

Example:

```
//--- example for CArrayLong::Shutdown()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   long  element      // Element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true if successful, false - if you can not add an element.

Example:

```
//--- example for CArrayLong::Add(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const long&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayLong::AddArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const CArrayLong*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayLong-source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayLong::AddArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayLong *src=new CArrayLong;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
```

```
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element in the array to the specified position.

```
bool  Insert(
   long  element,     // Element to insert
   int   pos          // Position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayLong::Insert(long,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array ukazannogy position.

```
bool  InsertArray(
   const long&  src[],     // Source array
   int           pos        // Position
   )
```

Parameters

src[]

[in] Reference to an array of source elements to insert

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayLong::InsertArray(const long &[],int)
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array ukazannogy position.

```
bool  InsertArray(
   CArrayLong*  src,      // Pointer to the source
   int           pos      // Position
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayLong-source elements to insert.

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayLong::InsertArray(const CArrayLong*,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayLong *src=new CArrayLong;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
```

```
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const long&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayLong::AssignArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const CArrayLong*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayLong-source element to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayLong::AssignArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayLong *src  =new CArrayLong;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
```

```
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified position array.

```
bool  Update(
   int   pos,          // Position
   long  element       // Value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value element

Return Value

true if successful, false - if you can not change the element.

Example:

```
//--- example for CArrayLong::Update(int,long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,1000000))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // Position
   int  shift       // Shift
   )
```

Parameters

pos

[in] Position of the moved element array

shift

[in] The value of displacement (both positive and negative).

Return Value

true if successful, false - if you can not move the item.

Example:

```
//--- example for CArrayLong::Shift(int,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the given position in the array.

```
bool  Delete(
   int  pos      // Position
   )
```

Parameters

pos

[in] Position removes the element in the array.

Return Value

true if successful, false - if you can not remove the element.

Example:

```
//--- example for CArrayLong::Delete(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from a given position in the array.

```
bool  DeleteRange(
   int  from,     // Position of the first element
   int  to        // Position of the last element
   )
```

Parameters

from

[in] Position of the first removes the element in the array.

to

[in] Position of the last deleted element in the array.

Return Value

true if successful, false - if you can not remove elements.

Example:

```
//--- example for CArrayLong::DeleteRange(int,int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the given position in the array.

```
long  At(
   int  pos      // Position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element in case of success, LONG_MAX if there was an attempt to get an element of not existing positions (the last error ERR_OUT_OF_RANGE).

Note

Of course, LONG_MAX may be a valid value of an array element, so having a value, always check the last error code.

Example:

```
//--- example for CArrayLong::At(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      long result=array.At(i);
      if(result==LONG_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- Error reading from the array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
```

```
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const long&  src[]      // Source array
   ) const
```

Parameters

src[]

[in] Reference to an array of source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayLong::CompareArray(const long &[])
#include <Arrays\ArrayLong.mqh>
//---
long src[];
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArrayconst

Compares array with another array.

```
bool  CompareArrayconst(
   const CArrayLong*  src      // Pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of class CArrayLong-source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayLong::CompareArray(const CArrayLong*)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayLong *src=new CArrayLong;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts element in a sorted array.

```
bool  InsertSort(
   long  element      // Element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayLong::InsertSort(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(1000000))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the model in sorted array.

```
int  Search(
   long  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayLong::Search(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(1000000)!=-1) printf("Element found");
   else                          printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element of larger sample, in sorted array.

```
int  SearchGreat(
   long  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayLong::SearchGreat(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(1000000)!=-1) printf("Element found");
   else                               printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element less than the sample in the sorted array.

```
int  SearchLess(
   long  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayLong::SearchLess(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(1000000)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element greater than or equal to the model in sorted array.

```
int  SearchGreatOrEqual(
   long  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayLong::SearchGreatOrEqual(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(1000000)!=-1) printf("Element found");
   else                                      printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element less than or equal to the model in sorted array.

```
int  SearchLessOrEqual(
   long  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayLong::SearchLessOrEqual(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(1000000)!=-1) printf("Element found");
   else                                     printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Finds the first element equal to the model in sorted array.

```
int  SearchFirst(
   long  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayLong::SearchFirst(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(1000000)!=-1) printf("Element found");
   else                               printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Finds the last element equal to the model in sorted array.

```
int  SearchLast(
   long  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayLong::SearchLast(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(1000000)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the model in the array.

```
int  SearchLinear(
   long  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the element was not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayLong::SearchLinear(long)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(1000000)!=-1) printf("Element found");
   else                                printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayLong::Save(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayLong *array=new CArrayLong;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

```
      }
   delete array;
  }
```

Load

Loads data array from a file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayLong::Load(int)
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayLong *array=new CArrayLong;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %I64",i,array.At(i));
```

```
      }
   delete array;
  }
```

Type

Gets the type identifier of the array.

```
virtual int  Type() const
```

Return Value

ID type of the array (for CArrayLong - 84).

Example:

```
//--- example for CArrayLong::Type()
#include <Arrays\ArrayLong.mqh>
//---
void OnStart()
  {
   CArrayLong *array=new CArrayLong;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayFloat

CArrayFloat class is a class of dynamic array of variables of type float.

Description

Class CArrayFloat provides the ability to work with a dynamic array of variables of type float.In the class implemented the ability to add / insert / delete elements in an array, sort array, searching in sorted array.In addition, the implemented methods of work with the file.

Declaration

```
class CArrayFloat : public CArray
```

Title

```
#include <Arrays\ArrayFloat.mqh>
```

Class Methods

Attributes	
Delta	Set the comparison tolerance
Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds to the end of the array elements from another array
AddArray	Adds to the end of the array elements from another array
Insert	Inserts an element in the array to the specified position
InsertArray	Inserts an array of elements from another array with the specified position
InsertArray	Inserts an array of elements from another array with the specified position
AssignArray	Copies the array elements from another array
AssignArray	Copies the array elements from another array
Update methods	

Update	Changes the element at the specified position array
Shift	Moves an item from a given position in the array to the specified offset
Delete	Removes the element from the specified position array
DeleteRange	Deletes a group of elements from the specified position array
Access methods	
At	Gets the element from the specified position array
Compare methods	
CompareArray	Compares array with another array
CompareArray	Compares array with another array
Sorted array operations	
InsertSort	Inserts element in a sorted array
Search	Searches for an element equal to the model in sorted array
SearchGreat	Searches for an element of more samples in sorted array
SearchLess	Searches for an element less than the sample in the sorted array
SearchGreatOrEqual	Searches for an element greater than or equal to the model in sorted array
SearchLessOrEqual	Searches for an element less than or equal to the model in sorted array
SearchFirst	Finds the first element equal to the model in sorted array
SearchLast	Finds the last element equal to the model in sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from a file
virtual Type	Gets the type identifier of the array

Delta

Set tolerance comparison.

```
void  Delta(
   float  delta      // Tolerance
   )
```

Parameters

delta

[in] the new value of the admission of comparison.

Return Value

None

Note

Admission of comparison used in the search. Values are considered equal if their difference is less than or equal to tolerance. The default tolerance is 0.0.

Example:

```
//--- example for CArrayFloat::Delta(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- set compare variation
   array.Delta(0.001);
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // Number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true if successful, false - if there was an attempt to seek the amount is less than or equal to zero, or if the array did not increase.

Note

To reduce memory fragmentation, increase the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayFloat::Reserve(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // Size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true if successful, false - if there was an attempt to set the size of less than or equal to zero.

Note

Changing the size of the array allows optimal use of memory.Superfluous elements on the right lost.To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayFloat::Resize(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true if successful, false - if an error occurred.

Example:

```
//--- example for CArrayFloat::Shutdown()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   float  element      // Element to add
   )
```

Parameters

element

[in] Value of the element to add to the array.

Return Value

true if successful, false - if you can not add an element.

Example:

```
//--- example for CArrayFloat::Add(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const float&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayFloat::AddArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const CArrayFloat*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayFloat-source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayFloat::AddArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayFloat *src=new CArrayFloat;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
```

```
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element in the array to the specified position.

```
bool  Insert(
   float  element,     // Element to insert
   int    pos          // Position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayFloat::Insert(float,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array with the specified position.

```
bool  InsertArray(
   const float&  src[],     // Source array
   int           pos        // Position
   )
```

Parameters

src[]

[in] Reference to an array of source elements to insert

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayFloat::InsertArray(const float &[],int)
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array specified position.

```
bool  InsertArray(
   CArrayFloat*  src,     // Pointer to the source
   int            pos      // Position
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayFloat-source elements to insert.

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayFloat::InsertArray(const CArrayFloat*,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayFloat *src=new CArrayFloat;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
```

```
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const float&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayFloat::AssignArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const CArrayFloat*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayFloat-source element to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayFloat::AssignArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayFloat *src  =new CArrayFloat;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
```

```
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified position array.

```
bool  Update(
   int    pos,          // Position
   float  element       // Value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value element

Return Value

true if successful, false - if you can not change the element.

Example:

```
//--- example for CArrayFloat::Update(int,float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,100.0))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // Position
   int  shift       // Shift
   )
```

Parameters

pos

[in] Position of the moved element array

shift

[in] The value of displacement (both positive and negative).

Return Value

true if successful, false - if you can not move the item.

Example:

```
//--- example for CArrayFloat::Shift(int,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the given position in the array.

```
bool  Delete(
   int  pos      // Position
   )
```

Parameters

pos

[in] Position removes the element in the array.

Return Value

true if successful, false - if you can not remove the element.

Example:

```
//--- example for CArrayFloat::Delete(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from a given position in the array.

```
bool  DeleteRange(
   int  from,     // Position of the first element
   int  to        // Position of last element
   )
```

Parameters

from

[in] Position of the first removes the element in the array.

to

[in] Position of the last deleted element in the array.

Return Value

true if successful, false - if you can not remove elements.

Example:

```
//--- example for CArrayFloat::DeleteRange(int,int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the given position in the array.

```
float  At(
   int  pos      // Position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element in case of success, FLT_MAX if there was an attempt to get an element of not existing positions (the last error ERR_OUT_OF_RANGE).

Note

Of course, FLT_MAX may be a valid value of an array element, so having a value, always check the last error code.

Example:

```
//--- example for CArrayFloat::At(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      float result=array.At(i);
      if(result==FLT_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- error reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
```

```
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const float&  src[]      // Source array
   ) const
```

Parameters

src[]

[in] Reference to an array of source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayFloat::CompareArray(const float &[])
#include <Arrays\ArrayFloat.mqh>
//---
float src[];
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

AssignArrayconst

Copies the array elements from another array.

```
bool  AssignArrayconst(
   const CArrayFloat*  src      // Pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of class CArrayFloat-source element to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayFloat::CompareArray(const CArrayFloat*)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayFloat *src=new CArrayFloat;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts element in a sorted array.

```
bool  InsertSort(
   float  element      // Element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayFloat::InsertSort(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(100.0))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   float  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayFloat::Search(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(100.0)!=-1) printf("Element found");
   else                        printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element of more samples in sorted array.

```
int  SearchGreat(
   float  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayFloat::SearchGreat(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(100.0)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element less than the sample in the sorted array.

```
int  SearchLess(
   float  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayFloat:: SearchLess(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(100.0)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element greater than or equal to the sample in the sorted array.

```
int  SearchGreatOrEqual(
   float  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayFloat::SearchGreatOrEqual(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(100.0)!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element less than or equal to the sample in the sorted array.

```
int  SearchLessOrEqual(
   float  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayFloat::SearchLessOrEqual(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(100.0)!=-1) printf("Element found");
   else                                   printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Finds the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   float  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayFloat::SearchFirst(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(100.0)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Finds the last element equal to the model in sorted array.

```
int  SearchLast(
   float  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayFloat::SearchLast(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(100.0)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   float  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the element was not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayFloat::SearchLinear(float)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(100.0)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayFloat::Save(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

```
      }
   delete array;
  }
```

Load

Loads data array from a file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayFloat::Load(int)
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %f",i,array.At(i));
```

```
      }
   delete array;
  }
```

Type

Gets the type identifier of the array.

```
virtual int  Type() const
```

Return Value

ID type of the array (for CArrayFloat - 87).

Example:

```
//--- example for CArrayFloat::Type()
#include <Arrays\ArrayFloat.mqh>
//---
void OnStart()
  {
   CArrayFloat *array=new CArrayFloat;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayDouble

CArrayDouble class is a class of dynamic array of variables of type double.

Description

Class CArrayDouble provides the ability to work with a dynamic array of variables of type double.In the class implemented the ability to add / insert / delete elements in an array, sort array, searching in sorted array.In addition, the implemented methods of work with the file.

Declaration

```
class CArrayDouble : public CArray
```

Title

```
#include <Arrays\ArrayDouble.mqh>
```

Class Methods

Attributes	
Delta	Set the comparison tolerance
Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a full memory release
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds to the end of the array elements from another array
AddArray	Adds to the end of the array elements from another array
Insert	Inserts an element in the array to the specified position
InsertArray	Inserts an array of elements from another array with the specified position
InsertArray	Inserts an array of elements from another array with the specified position
AssignArray	Copies the array elements from another array
AssignArray	Copies the array elements from another array
Update methods	

Update	Changes the element at the specified position array
Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified position array
DeleteRange	Deletes a group of elements from the specified position array
Access methods	
At	Gets the element from the specified position array
Compare methods	
CompareArray	Compares array with another array
CompareArray	Compares array with another array
Search for min/max	
Minimum	Gets index of the lowest element of the array in the specified range
Maximum	Gets index of the highest element of the array in the specified range
Sorted array operations	
InsertSort	Inserts element in a sorted array
Search	Searches for an element equal to the model in sorted array
SearchGreat	Searches for an element of more samples in sorted array
SearchLess	Searches for an element less than the sample in the sorted array
SearchGreatOrEqual	Searches for an element greater than or equal to the model in sorted array
SearchLessOrEqual	Searches for an element less than or equal to the model in sorted array
SearchFirst	Finds the first element equal to the model in sorted array
SearchLast	Finds the last element equal to the model in sorted array
SearchLinear	Searches for the element equal to the sample in the array

Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from a file
virtual Type	Gets the type identifier of the array

Derived classes:

- CIndicatorBuffer

Delta

Set tolerance comparison.

```
void  Delta(
   double  delta      // Tolerance
   )
```

Parameters

delta

[in] The new value of the admission of comparison.

Return Value

No

Note

Admission of comparison used in the search. Values are considered equal if their difference is less than or equal to tolerance. The default tolerance is 0.0.

Example:

```
//--- example for CArrayDouble::Delta(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- set compare variation
   array.Delta(0.001);
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // Number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true if successful, false - if there was an attempt to seek the amount is less than or equal to zero, or if the array did not increase.

Note

To reduce memory fragmentation, increase the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayDouble::Reserve(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // Size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true if successful, false - if there was an attempt to set the size of less than or equal to zero.

Note

Changing the size of the array allows optimal use of memory.Superfluous elements on the right lost.To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayDouble::Resize(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true if successful, false - if an error occurred.

Example:

```
//--- example for CArrayDouble::Shutdown()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   double  element      // Element to add
   )
```

Parameters

element

[in] value of the element to add to the array.

Return Value

true if successful, false - if you can not add an element.

Example:

```
//--- example for CArrayDouble::Add(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(i))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const double&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayDouble::AddArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const CArrayDouble*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayDouble-source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayDouble::AddArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayDouble *src=new CArrayDouble;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
```

```
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element in the array to the specified position.

```
bool  Insert(
   double  element,     // Element to insert
   int     pos          // Position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayDouble::Insert(double,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(i,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array specified position.

```
bool  InsertArray(
   const double&  src[],      // Source array
   int            pos         // Position
   )
```

Parameters

src[]

[in] Reference to an array of source elements to insert

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayDouble::InsertArray(const double &[],int)
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array specified position.

```
bool  InsertArray(
   CArrayDouble*  src,     // Pointer to the source
   int            pos      // Position
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayDouble-source elements to insert.

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayDouble::InsertArray(const CArrayDouble*,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayDouble *src=new CArrayDouble;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
```

```
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const double&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayDouble::AssignArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const CArrayDouble*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayDouble-source element to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayDouble::AssignArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayDouble *src  =new CArrayDouble;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
```

```
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified position array.

```
bool  Update(
   int     pos,          // Position
   double  element       // Value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value.

Return Value

true if successful, false - if you can not change the element.

Example:

```
//--- example for CArrayDouble::Update(int,double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,100.0))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // Position
   int  shift       // Shift
   )
```

Parameters

pos

[in] Position of the moved element array

shift

[in] The value of displacement (both positive and negative).

Return Value

true if successful, false - if you can not move the item.

Example:

```
//--- example for CArrayDouble::Shift(int,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the given position in the array.

```
bool  Delete(
   int  pos      // Position
   )
```

Parameters

pos

[in] position removes the element in the array.

Return Value

true if successful, false - if you can not remove the element.

Example:

```
//--- example for CArrayDouble::Delete(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from a given position in the array.

```
bool  DeleteRange(
   int  from,     // Position of the first element
   int  to        // Position of last element
   )
```

Parameters

from

[in] Position of the first removes the element in the array.

to

[in] Position of the last deleted element in the array.

Return Value

true if successful, false - if you can not remove elements.

Example:

```
//--- example for CArrayDouble::DeleteRange(int,int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the given position in the array.

```
double  At(
   int  pos      // Position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element in case of success, DBL_MAX if there was an attempt to get an element of not existing positions (the last error ERR_OUT_OF_RANGE).

Note

Of course, DBL_MAX may be a valid value of an array element, so having a value, always check the last error code.

Example:

```
//--- example for CArrayDouble::At(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      double result=array.At(i);
      if(result==DBL_MAX && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //---  Error reading from the array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
```

```
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const double&  src[]      // Source array
   ) const
```

Parameters

src[]

[in] Reference to an array of source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayDouble::CompareArray(const double &[])
#include <Arrays\ArrayDouble.mqh>
//---
double src[];
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const CArrayDouble*  src      // Pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of class CArrayDouble-source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayDouble::CompareArray(const CArrayDouble*)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayDouble *src=new CArrayDouble;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

Minimum

Gets index of the lowest element of the array in the specified range.

```
int  Minimum(
   int  start,      // starting index
   int  count       // number of elements to proceed
   ) const
```

Parameters

start

[in] Starting index of the array.

count

[in] Number of elements to proceed.

Returned value

Index of the lowest element of the array in the specified range.

Maximum

Gets index of the highest element of the array in the specified range.

```
int  Maximum(
   int  start,      // starting index
   int  count       // number of elements to proceed
   ) const
```

Parameters

start

[in] Starting index of the array.

count

[in] Number of elements to proceed.

Returned value

Index of the highest element of the array in the specified range.

InsertSort

Inserts element in a sorted array.

```
bool  InsertSort(
   double  element      // Element to insert
   )
```

Parameters

element

[in] value of the element to be inserted into a sorted array

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayDouble::InsertSort(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(100.0))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   double  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayDouble::Search(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search(100.0)!=-1) printf("Element found");
   else                        printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element of more samples in sorted array.

```
int  SearchGreat(
   double  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayDouble::SearchGreat(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat(100.0)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element less than the sample in the sorted array.

```
int  SearchLess(
   double  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayDouble:: SearchLess(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess(100.0)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element greater than or equal to the sample in the sorted array.

```
int  SearchGreatOrEqual(
   double  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayDouble::SearchGreatOrEqual(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual(100.0)!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element less than or equal to the sample in the sorted array.

```
int  SearchLessOrEqual(
   double  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayDouble::SearchLessOrEqual(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual(100.0)!=-1) printf("Element found");
   else                                   printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Finds the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   double  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayDouble::SearchFirst(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst(100.0)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Finds the last element equal to the model in sorted array.

```
int  SearchLast(
   double  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayDouble::SearchLast(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast(100.0)!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the sorted array.

```
int  SearchLinear(
   double  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the element was not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayDouble::SearchLinear(double)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear(100.0)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayDouble::Save(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(i);
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

```
      }
   //--- delete array
   delete array;
  }
```

Load

Loads data array from a file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayDouble::Load(int)
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = %f",i,array.At(i));
```

```
      }
   //--- delete array
   delete array;
  }
```

Type

Gets the type identifier of the array.

```
virtual int  Type() const
```

Return Value

ID type of the array (for CArrayDouble - 87).

Example:

```
//--- example for CArrayDouble::Type()
#include <Arrays\ArrayDouble.mqh>
//---
void OnStart()
  {
   CArrayDouble *array=new CArrayDouble;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayString

CArrayString class is a class of dynamic array of variables of type string.

Description

Class CArrayString provides the ability to work with a dynamic array of variables of type string.In the class implemented the ability to add / insert / delete elements in an array, sort array, searching in sorted array.In addition, the implemented methods of work with the file.

Declaration

```
class CArrayString : public CArray
```

Title

```
#include <Arrays\ArrayString.mqh>
```

Class Methods

Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Sets a new (smaller) size of the array
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds to the end of the array elements from another array
AddArray	Adds to the end of the array elements from another array
Insert	Inserts an element in the array to the specified position
InsertArray	Inserts an array of elements from another array with the specified position
InsertArray	Inserts an array of elements from another array with the specified position
AssignArray	Copies the array elements from another array
AssignArray	Copies the array elements from another array
Update methods	
Update	Changes the element at the specified position array

Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Delete	Removes the element from the specified position array
DeleteRange	Deletes a group of elements from the specified position array
Access methods	
At	Gets the element from the specified position array
Compare methods	
CompareArray	Compares array with another array
CompareArray	Compares array with another array
Sorted array opetations	
InsertSort	Inserts element in a sorted array
Search	Searches for an element equal to the sample in sorted array
SearchGreat	Searches for an element of more samples in sorted array
SearchLess	Searches for an element less than the sample in the sorted array
SearchGreatOrEqual	Searches for an element greater than or equal to the sample in sorted array
SearchLessOrEqual	Searches for an element less than or equal to the sample in sorted array
SearchFirst	Finds the first element equal to the sample in sorted array
SearchLast	Finds the last element equal to the sample in sorted array
SearchLinear	Searches for the element equal to the sample in the array
Input/output	
virtual Save	Saves data array in the file
virtual Load	Loads data array from a file
virtual Type	Gets the type identifier of the array

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // Number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true if successful, false - if there was an attempt to seek the amount is less than or equal to zero, or if the array did not increase.

Note

To reduce memory fragmentation, increase the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayString::Reserve(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reserve memory
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // Size
   )
```

Parameters

size

[in] New size of the array.

Return Value

true if successful, false - if there was an attempt to set the size of less than or equal to zero.

Note

Changing the size of the array allows optimal use of memory.Superfluous elements on the right lost.To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayString::Resize(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a full memory release.

```
bool  Shutdown()
```

Return Value

true if successful, false - if an error occurred.

Example:

```
//--- example for CArrayString::Shutdown()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Add

Adds an element to the end of the array.

```
bool  Add(
   string  element      // Element to add
   )
```

Parameters

element

[in] value of the element to add to the array.

Return Value

true if successful, false - if you can not add an element.

Example:

```
//--- example for CArrayString::Add(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(IntegerToString(i)))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const string&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayString::AddArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const CArrayString*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayString - the source of elements to add.

Return Value

true if successful, false - if you can not add items.

Example:

```
//--- example for CArrayString::AddArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayString *src=new CArrayString;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array
   delete src;
```

```
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element in the array to the specified position.

```
bool  Insert(
   string  element,     // Element to insert
   int     pos          // Position
   )
```

Parameters

element

[in] Value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayString::Insert(string,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(IntegerToString(i),0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array specified position.

```
bool  InsertArray(
   const string&  src[],     // Source array
   int             pos        // Position
   )
```

Parameters

src[]

[in] Reference to an array of source elements to insert

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayString::InsertArray(const string &[],int)
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array specified position.

```
bool  InsertArray(
   CArrayString*  src,     // Pointer to the source
   int             pos      // Position
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayString - the source of elements to insert.

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Example:

```
//--- example for CArrayString::InsertArray(const CArrayString*,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayString *src=new CArrayString;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- insert another array
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
```

```
      return;
     }
   //--- delete source array
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const string&  src[]      // Source array
   )
```

Parameters

src[]

[in] Reference to an array of source elements to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayString::AssignArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const CArrayString*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayString - source of elements to copy.

Return Value

true if successful, false - if you can not copy the items.

Example:

```
//--- example for CArrayString::AssignArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayString *src  =new CArrayString;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array
```

```
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified position array.

```
bool  Update(
   int     pos,          // Position
   string  element       // Value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value element

Return Value

true if successful, false - if you can not change the element.

Example:

```
//--- example for CArrayString::Update(int, string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,"ABC"))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // Position
   int  shift       // Shift
   )
```

Parameters

pos

[in] Position of the moved element array

shift

[in] The value of displacement (both positive and negative).

Return Value

true if successful, false - if you can not move the item.

Example:

```
//--- example for CArrayString::Shift(int,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the given position in the array.

```
bool  Delete(
   int  pos      // Position
   )
```

Parameters

pos

[in] position removes the element in the array.

Return Value

true if successful, false - if you can not remove the element.

Example:

```
//--- example for CArrayString::Delete(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete element
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from a given position in the array.

```
bool  DeleteRange(
   int  from,     // Position of the first element
   int  to        // Position of last element
   )
```

Parameters

from

[in] Position of the first removes the element in the array.

to

[in] Position of the last deleted element in the array.

Return Value

true if successful, false - if you can not remove elements.

Example:

```
//--- example for CArrayString::DeleteRange(int,int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the given position in the array.

```
string  At(
   int  pos      // Position
   ) const
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element in case of success, "- if there was an attempt to get an element of not existing positions (the last error ERR_OUT_OF_RANGE).

Note

Of course, "" may be a valid value of an array element, so having a value, always check the last error code.

Example:

```
//--- example for CArrayString::At(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      string result=array.At(i);
      if(result=="" && GetLastError()==ERR_OUT_OF_RANGE)
        {
         //--- Error reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
```

```
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const string&  src[]      // Source array
   ) const
```

Parameters

src[]

[in] Reference to an array of source elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayString::CompareArray(const string &[])
#include <Arrays\ArrayString.mqh>
//---
string src[];
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete array
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArrays(
   const CArrayString*  src      // Pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of class CArrayString - the source of elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayString::CompareArray(const CArrayString*)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayString *src=new CArrayString;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- add source arrays elements
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts element in a sorted array.

```
bool  InsertSort(
   string  element      // Element to insert
   )
```

Parameters

element

[in] value of the element to be inserted into a sorted array

Return Value

true if successful, false - if you can not insert the element.

Example:

```
//--- example for CArrayString::InsertSort(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort("ABC"))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   string  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayString::Search(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.Search("ABC")!=-1) printf("Element found");
   else                        printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element of more samples in sorted array.

```
int  SearchGreat(
   string  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayString::SearchGreat(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreat("ABC")!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element less than the sample in the sorted array.

```
int  SearchLess(
   string  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayString:: SearchLess(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLess("ABC")!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element greater than or equal to the sample in the sorted array.

```
int  SearchGreatOrEqual(
   string  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayString:: SearchGreatOrEqual(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchGreatOrEqual("ABC")!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element less than or equal to the sample in the sorted array.

```
int  SearchLessOrEqual(
   string  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayString:: SearchLessOrEqual(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLessOrEqual("ABC")!=-1) printf("Element found");
   else                                   printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Finds the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   string  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayString:: SearchFirst(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchFirst("ABC")!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Finds the last element equal to the model in sorted array.

```
int  SearchLast(
   string  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayString:: SearchLast(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- search element
   if(array.SearchLast("ABC")!=-1) printf("Element found");
   else                            printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLinear

Searches for the element equal to the sample in the array.

```
int  SearchLinear(
   string  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the element was not found.

Note

The method uses the linear search (or sequential search) algorithm for unsorted arrays.

Example:

```
//--- example for CArrayString::SearchLinear(string)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- search element
   if(array.SearchLinear("ABC")!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayString::Save(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayString *array=new CArrayString;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      array.Add(IntegerToString(i));
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
```

```
      }
   delete array;
  }
```

Load

Loads data array from a file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayString::Load(int)
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CArrayString *array=new CArrayString;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use arrays elements
   for(int i=0;i<array.Total();i++)
     {
      printf("Element[%d] = '%s'",i,array.At(i));
```

```
      }
   delete array;
  }
```

Type

Gets the type identifier of the array.

```
virtual int  Type() const
```

Return Value

ID type of the array (for CArrayString - 89).

Example:

```
//--- example for CArrayString::Type()
#include <Arrays\ArrayString.mqh>
//---
void OnStart()
  {
   CArrayString *array=new CArrayString;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CArrayObj

CArrayObj class is a class of dynamic array of pointers to instances of CObject and his heirs.

Description

Class CArrayObj provides the ability to work with a dynamic array of pointers to instances of CObject and his heirs. This gives the possibility to work as a multidimensional dynamic arrays of primitive data types, and the more difficult for organized data structures.

In the class implemented the ability to add / insert / delete elements in an array, sort array, searching in sorted array. In addition, the implemented methods of work with the file.

There are certain subtleties of the class CArrayObj.

Declaration

```
class CArrayObj : public CArray
```

Title

```
#include <Arrays\ArrayObj.mqh>
```

Class Method

Attributes	
FreeMode	Gets the flag memory management
FreeMode	Sets the flag memory management
Memory control	
Reserve	Allocates memory to increase the size of the array
Resize	Sets a new (smaller) size of the array
Shutdown	Clears the array with a total exemption memory array (not element).
Add methods	
Add	Adds an element to the end of the array
AddArray	Adds an element to the end of the array
Insert	Inserts an element in the array to the specified position
InsertArray	Inserts an array of elements from another array with the specified position
AssignArray	Copies the array elements from another array
Update methods	

Update	Changes the element at the specified position array
Shift	Moves an item from a given position in the array to the specified offset
Delete methods	
Detach	Gets the element from the specified position and removing it from the array
Delete	Removes the element from the specified position array
DeleteRange	Deletes a group of elements from the specified position array
Clear	Removes all elements of the array without the release of the memory array
Access methods	
At	Gets the element from the specified position array
Compare methods	
CompareArray	Compares array with another array
Sorted array operations	
InsertSort	Inserts element in a sorted array
Search	Searches for an element equal to the model in sorted array
SearchGreat	Searches for an element of more samples in sorted array
SearchLess	Searches for an element less than the sample in the sorted array
SearchGreatOrEqual	Searches for an element greater than or equal to the model in sorted array
SearchLessOrEqual	Searches for an element less than or equal to the model in sorted array
SearchFirst	Finds the first element equal to the model in sorted array
SearchLast	Finds the last element equal to the model in sorted array
Input/output	
Save	Saves data array in the file
Load	Loads data array from a file

Type	Gets the type identifier of the array

Derived classes:

- CIndicator
- CIndicators

Practical application of arrays are descendants of class CObject (including all classes of the standard library).

For example, consider the options for two-dimensional array:

```
#include <Arrays\ArrayDouble.mqh>
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int i,j;
   int first_size=10;
   int second_size=100;
//--- create array
   CArrayObj    *array=new CArrayObj;
   CArrayDouble *sub_array;
//---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
//--- create subarrays
   for(i=0;i<first_size;i++)
     {
      sub_array=new CArrayDouble;
      if(sub_array==NULL)
        {
         delete array;
         printf("Object create error");
         return;
        }
      //--- fill array
      for(j=0;j<second_size;j++)
        {
         sub_array.Add(i*j);
        }
      array.Add(sub_array);
     }
//--- create array OK
   for(i=0;i<first_size;i++)
     {
```

```
      sub_array=array.At(i);
      for(j=0;j<second_size;j++)
        {
         double element=sub_array.At(j);
         //--- use array element
        }
     }
   delete array;
  }
```

Subtleties

The class has a mechanism to control dynamic memory, so be careful when working with elements of the array.

Mechanism of memory management can be switched on / off using the method FreeMode (bool). By default, the mechanism is enabled.

Accordingly, there are two options for dealing with the class CArrayObj:

1. Mechanism of memory management is enabled. (default)

In this case, CArrayObj take responsibility for freeing the memory elements after their removal from the array. In this program the user should not free the array elements.

Example:

```
   int i;
//--- Create an array
   CArrayObj *array=new CArrayObj;
//--- Fill array elements
   for(i=0;i<10;i++) array.Add(new CObject);
//--- Do something
   for(i=0;i<array.Total();i++)
     {
      CObject *object=array.At(i);
      //--- Action with an element
      . . .
     }
//--- Remove the array with the elements
   delete array;
```

2. Mechanism of memory management is turned off.

In this case, CArrayObj not otvetstvechaet for freeing the memory elements after their removal from the array. In this program the user must free the array elements.

Example:

```
   int i;
```

```
//--- Create an array
   CArrayObj *array=new CArrayObj;
//--- Disable the mechanism of memory management
   array.FreeMode(false);
//--- Fill array elements
   for(i=0;i<10;i++) array.Add(new CObject);
//--- Do something
   for(i=0;i<array.Total();i++)
     {
      CObject *object=array.At(i);
      //--- Action with an element
      . . .
     }
//--- Remove array elements
   while(array.Total()) delete array.Detach();
//--- Remove empty array
   delete array;
```

FreeMode

Gets the flag memory management.

```
bool  FreeMode() const
```

Return Value

Flag of memory management.

Example:

```
//--- example for CArrayObj::FreeMode()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get free mode flag
   bool array_free_mode=array.FreeMode();
   //--- delete array
   delete array;
  }
```

FreeMode

Sets the flag memory management.

```
void  FreeMode(
   bool  mode      // New flag
   )
```

Parameters

mode

[in] New value of the flag memory management.

Return Value

None.

Note

Setting the flag memory management - an important point in the use of class CArrayObj.Since the array elements are pointers to dynamic objects, it is important to determine what to do with them when removing from the array.

If the flag is set, removing an element from the array, the element is automatically deleted by the operator delete.If the flag is not set, it is assumed that a pointer to the deleted object is still somewhere in the user program and will be relieved of it (the program) then.

If the user resets the flag memory management, the user must understand their responsibility for the removal of the array before the completion of the program, because otherwise, is not freed memory occupied by the elements when they create new operator.

When large amounts of data, it could lead, eventually, even to break your terminal.If the user does not reset the flag memory management, there is another "reef".

Using pointers, array, stored somewhere in the local variables, after removing the array will lead to a critical error and crashes the program user.By default, the memory management flag is set, ie the class of the array is responsible for freeing the memory elements.

Example:

```
//--- example for CArrayObj::FreeMode(bool)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reset free mode flag
   array.FreeMode(false);
```

```
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Reserve

Allocates memory to increase the size of the array.

```
bool  Reserve(
   int  size      // Number
   )
```

Parameters

size

[in] The number of additional elements of the array.

Return Value

true if successful, false - if there was an attempt to seek the amount is less than or equal to zero, or if the array did not increase.

Note

To reduce memory fragmentation, increase the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayObj::Reserve(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   if(!array.Reserve(1024))
     {
      printf("Reserve error");
      delete array;
      return;
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Resize

Sets a new (smaller) size of the array.

```
bool  Resize(
   int  size      // Size
   )
```

Parameters

size

[in] New size of the array.

Return Value

To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Note

Changing the size of array allows to use memory in the optimal way. Excess element located to the right are lost. The memory for lost elements is released or not depending on the mode of memory management.

To reduce fragmentation of memory, change the size of the array is made with a step previously given through the method of Step (int), or 16 (default).

Example:

```
//--- example for CArrayObj::Resize(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- resize array
   if(!array.Resize(10))
     {
      printf("Resize error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
```

```
}
```

Clear

Removes all elements of the array without the release of the memory array.

```
void  Clear()
```

Return Value

No.

Note

If enabled memory management, memory, deleted items are exempt.

Example:

```
//--- example for CArrayObj::Clear()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- clear array
   array.Clear();
   //--- delete array
   delete array;
  }
```

Shutdown

Clears the array with a total exemption memory array (not element).

```
bool  Shutdown()
```

Return Value

true if successful, false - if an error occurred.

Note

If enabled memory management, memory, deleted items are exempt.

Example:

```
//--- example for CArrayObj::Shutdown()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shutdown array
   if(!array.Shutdown())
     {
      printf("Shutdown error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

CreateElement

Creates a new array element at the specified position.

```
bool  CreateElement(
   int  index      // Position
   )
```

Parameters

index

[in] position in which you want to create a new element.

Return Value

true if successful, false - if you can not create element.

Note

Method CreateElement (int) in class CArrayObj always returns false and does not perform any action. If necessary, in a derived class, method CreateElement (int) should be implemented.

Example:

```
//--- example for CArrayObj::CreateElement(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int        size=100;
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- fill array
   array.Reserve(size);
   for(int i=0;i<size;i++)
     {
      if(!array.CreateElement(i))
        {
         printf("Element create error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
```

```
}
```

Add

Adds an element to the end of the array.

```
bool  Add(
   CObject*  element      // Element to add
   )
```

Parameters

element

[in] value of the element to add to the array.

Return Value

true if successful, false - if you can not add an element.

Note

Element is not added to the array if the value for transmit invalid pointer (such as NULL).

Example:

```
//--- example for CArrayObj::Add(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 arrays elements
   for(int i=0;i<100;i++)
     {
      if(!array.Add(new CObject))
        {
         printf("Element addition error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AddArray

Adds to the end of the array elements from another array.

```
bool  AddArray(
   const CArrayObj *  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayDouble - source of elements to add.

Return Value

true if successful, false - if you can not add items.

Note

Adding elements of the array in the array is actually copying the pointers.Therefore, when calling the method, there is a "reef" - that there may be a pointer to a dynamic object in more than one variable.

```
//--- example
extern bool       make_error;
extern int        error;
extern CArrayObj *src;
//--- Create a new instance CArrayObj
//--- Default memory management is turned on
CArrayObj *array=new CArrayObj;
//--- Add (copy) the elements of the array-istochika
if(array!=NULL)
   bool result=array.AddArray(src);
if(make_error)
  {
   //--- Commit wrongdoing
   switch(error)
     {
      case 0:
         //--- Remove the array-source, without checking its flag memory management
         delete src;
         //--- Result:
         //--- May appeal
         //--- For "bitomu" pointer in the array receiver
         break;
      case 1:
         //--- Disable the mechanism of memory management in an array of source
         if(src.FreeMode()) src.FreeMode(false);
         //--- But do not remove the array-source
         //--- Result:
         //--- After removal of the array, the receiver may appeal
```

```
         //--- For "bitomu" pointer in the array-source
         break;
      case 2:
         //--- Disable the mechanism of memory management in an array of source
         src.FreeMode(false);
         //--- Disable the mechanism of memory management in an array receiver
         array.FreeMode(false);
         //--- Result:
         //--- After the completion of the program, obtain a "memory leak"
         break;
     }
  }
else
  {
   //--- Disable the mechanism of memory management in an array of source
   if(src.FreeMode()) src.FreeMode(false);
   //--- Remove the array-source
   delete src;
   //--- Result:
   //--- Treatment for an array-recepient be correct
   //--- Delete the array, the receiver will delete its elements
  }
```

Example:

```
//--- example for CArrayObj::AddArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- reset free mode flag
   src.FreeMode(false);
   //--- fill source array
```

```
   //--- . . .
   //--- add another array
   if(!array.AddArray(src))
     {
      printf("Array addition error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array without elements
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Insert

Inserts an element in the array to the specified position.

```
bool  Insert(
   CObject*  element,     // Element to insert
   int       pos          // Position
   )
```

Parameters

element

[in] value of the element to be inserted into an array

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not insert the element.

Note

Element is not added to the array if the value for transmit invalid pointer (such as NULL).

Example:

```
//--- example for CArrayObj::Insert(CObject*,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert elements
   for(int i=0;i<100;i++)
     {
      if(!array.Insert(new CObject,0))
        {
         printf("Insert error");
         delete array;
         return;
        }
     }
   //--- use array
   //--- . . .
   //--- delete array
```

```
   delete array;
  }
```

InsertArray

Inserts an array of elements from another array specified position.

```
bool  InsertArray(
   const CArrayObj*  src,     // Pointer to the source
   int               pos      // Position
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayObj-source elements to insert.

pos

[in] Position in the array to insert

Return Value

true if successful, false - if you can not paste items.

Note

See: CArrayObj::AddArray(const CArrayObj*).

Example:

```
//--- example for CArrayObj::InsertArray(const CArrayObj*,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- reset free mode flag
   src.FreeMode(false);
   //--- fill source array
   //--- . . .
   //--- insert another array
```

```
   if(!array.InsertArray(src,0))
     {
      printf("Array inserting error");
      delete src;
      delete array;
      return;
     }
   //--- delete source array without elements
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

AssignArray

Copies the array elements from another array.

```
bool  AssignArray(
   const CArrayObj*  src      // Pointer to the source
   )
```

Parameters

src

[in] Pointer to an instance of class CArrayObj - source of elements to copy.

Return Value

true if successful, false - if you can not copy the items.

Note

If the challenge AssignArray array receiver is not empty, all its elements will be removed from the array and, if the flag memory management, memory, deleted items will be released.Array-receiver is an exact copy of the array source. Additionally see CArrayObj::AddArray(const CArrayObj*).

Example:

```
//--- example for CArrayObj::AssignArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- reset free mode flag
   src.FreeMode(false);
   //--- fill source array
   //--- . . .
   //--- assign another array
   if(!array.AssignArray(src))
     {
```

```
      printf("Array assigned error");
      delete src;
      delete array;
      return;
     }
   //--- arrays is identical
   //--- delete source array without elements
   delete src;
   //--- use array
   //--- . . .
   //--- delete array
   delete array;
  }
```

Update

Changes the element at the specified position array.

```
bool  Update(
   int       pos,          // Position
   CObject*  element       // Value
   )
```

Parameters

pos

[in] Position of the element in the array to change

element

[in] New value element

Return Value

true if successful, false - if you can not change the element.

Note

The element does not change if we as a parameter to pass an invalid pointer (ie NULL). If enabled memory management, memory placeholder released.

Example:

```
//--- example for CArrayObj::Update(int,CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- update element
   if(!array.Update(0,new CObject))
     {
      printf("Update error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Shift

Moves an item from a given position in the array to the specified offset.

```
bool  Shift(
   int  pos,        // Position
   int  shift       // Shift
   )
```

Parameters

pos

[in] Position of the moved element array

shift

[in] The value of displacement (both positive and negative).

Return Value

true if successful, false - if you can not move the item.

Example:

```
//--- example for CArrayObj::Shift(int,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- shift element
   if(!array.Shift(10,-5))
     {
      printf("Shift error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Detach

Remove an item from a given position in the array.

```
CObject*  Detach(
   int  pos      // Position
   )
```

Parameters

pos

[in] Position of the seized item in the array.

Return Value

Pointer to the removal of elements in case of success, NULL - if you can not remove the element.

Note

When removed from the array element is not removed in any state of the flag memory management. Pointer to the array element withdrawn from the ingredients of the release after use.

Example:

```
//--- example for CArrayObj::Detach(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   CObject *object=array.Detach(0);
   if(object==NULL)
     {
      printf("Detach error");
      delete array;
      return;
     }
   //--- use element
   //--- . . .
   //--- delete element
   delete object;
   //--- delete array
   delete array;
  }
```

Delete

Removes the element from the given position in the array.

```
bool  Delete(
   int  pos      // Position
   )
```

Parameters

pos

[in] position removes the element in the array.

Return Value

true if successful, false - if you can not remove the element.

Note

If enabled memory management, memory, deleted items are exempt.

Example:

```
//--- example for CArrayObj::Delete(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   if(!array.Delete(0))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

DeleteRange

Deletes a group of elements from a given position in the array.

```
bool  DeleteRange(
   int  from,     // Position of the first element
   int  to        // Position of last element
   )
```

Parameters

from

[in] Position of the first removes the element in the array.

to

[in] Position of the last deleted element in the array.

Return Value

true if successful, false - if you can not remove elements.

Note

If enabled memory management, memory, deleted items are exempt.

Example:

```
//--- example for CArrayObj::DeleteRange(int,int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- delete elements
   if(!array.DeleteRange(0,10))
     {
      printf("Delete error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

At

Gets the element from the given position in the array.

```
CObject*  At(
   int  pos      // Position
   )
```

Parameters

pos

[in] Position of the desired element in the array.

Return Value

The value of the element, if successful, NULL-if there was an attempt to get an element of non-existent position.

Example:

```
//--- example for CArrayObj::At(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add elements
   //--- . . .
   for(int i=0;i<array.Total();i++)
     {
      CObject *result=array.At(i);
      if(result==NULL)
        {
         //--- Error reading from array
         printf("Get element error");
         delete array;
         return;
        }
      //--- use element
      //--- . . .
     }
   delete array;
  }
```

CompareArray

Compares array with another array.

```
bool  CompareArray(
   const CArrayObj*  src      // Pointer to the source
   ) const
```

Parameters

src

[in] Pointer to an instance of class CArrayObj - the source of elements for comparison.

Return Value

true if arrays are equal, false - if not.

Example:

```
//--- example for CArrayObj::CompareArray(const CArrayObj*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source array
   CArrayObj *src=new CArrayObj;
   if(src==NULL)
     {
      printf("Object create error");
      delete array;
      return;
     }
   //--- fill source array
   //--- . . .
   //--- compare with another array
   int result=array.CompareArray(src);
   //--- delete arrays
   delete src;
   delete array;
  }
```

InsertSort

Inserts element in a sorted array.

```
bool  InsertSort(
   CObject*  element      // Element to insert
   )
```

Parameters

element

[in] Value of the element to be inserted into a sorted array

Return Value

true if successful, false - if you can not insert the element.

Note

Element is not added to the array if the value for transmit invalid pointer (such as NULL).

Example:

```
//--- example for CArrayObj::InsertSort(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- insert element
   if(!array.InsertSort(new CObject))
     {
      printf("Insert error");
      delete array;
      return;
     }
   //--- delete array
   delete array;
  }
```

Search

Searches for an element equal to the sample in the sorted array.

```
int  Search(
   CObject*  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayObj::Search(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.Search(sample)!=-1) printf("Element found");
   else                         printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreat

Searches for an element of more samples in sorted array.

```
int  SearchGreat(
   CObject*  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayObj::SearchGreat(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchGreat(sample)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLess

Searches for an element less than the sample in the sorted array.

```
int  SearchLess(
   CObject*  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayObj:: SearchLess(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchLess(sample)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchGreatOrEqual

Searches for an element greater than or equal to the sample in the sorted array.

```
int  SearchGreatOrEqual(
   CObject*  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayObj::SearchGreatOrEqual(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchGreatOrEqual(sample)!=-1) printf("Element found");
   else                                     printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLessOrEqual

Searches for an element less than or equal to the sample in the sorted array.

```
int  SearchLessOrEqual(
   CObject*  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayObj:: SearchLessOrEqual(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchLessOrEqual(sample)!=-1) printf("Element found");
   else                                    printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchFirst

Finds the first element equal to the sample in the sorted array.

```
int  SearchFirst(
   CObject*  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayObj::SearchFirst(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchFirst(sample)!=-1) printf("Element found");
   else                              printf("Element not found");
   //--- delete array
   delete array;
  }
```

SearchLast

Finds the last element equal to the model in sorted array.

```
int  SearchLast(
   CObject*  element      // Sample
   ) const
```

Parameters

element

[in] The sample element to search in the array.

Return Value

The position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CArrayObj:: SearchLast(CObject*)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- sort array
   array.Sort();
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete array;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(array.SearchLast(sample)!=-1) printf("Element found");
   else                             printf("Element not found");
   //--- delete array
   delete array;
  }
```

Save

Saves data array in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to previously opened by FileOpen (...) function binary file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CArrayObj::Save(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int         file_handle;
   CArrayObj *array=new CArrayObj;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add arrays elements
   //--- . . .
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   delete array;
  }
```

Load

Loads data array from a file.s

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle to open earlier, with the function FileOpen (...), binary file.

Return Value

true - if successfully completed, false - if an error.

Note

When reading from the file array to create each element of the method is called CArrayObj::CreateElement(int).

Example:

```
//--- example for CArrayObj::Load(int)
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   int        file_handle;
   CArrayObj *array=new CArrayObj;
   //---
   if(array!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!array.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete array;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
```

```
   //--- use arrays elements
   //--- . . .
   //--- delete array
   delete array;
  }
```

Type

Gets the type identifier of the array.

```
virtual int  Type() const
```

Return Value

ID type of the array (for CArrayObj - 7778).

Example:

```
//--- example for CArrayObj::Type()
#include <Arrays\ArrayObj.mqh>
//---
void OnStart()
  {
   CArrayObj *array=new CArrayObj;
   //---
   if(array==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get array type
   int type=array.Type();
   //--- delete array
   delete array;
  }
```

CList

CList Class is a class of dynamic list of instances of the class CObject and his heirs.

Description

Class CList provides the ability to work with a list of instances of CObject and his heirs. In the class implemented the ability to add / insert / delete items in the list, sort the list, search in sorted list. In addition, the implemented methods of work with the file.

There are some subtleties of working with the class CList. The class has a mechanism to control dynamic memory, so be careful when working with elements of the list.

Subtleties of the mechanism of memory management similar to those described in CArrayObj.

Declaration

```
class CList : public CObject
```

Title

```
#include <Arrays\List.mqh>
```

Class Methods

Attributes	
FreeMode	Gets the flag memory management when deleting list items.
FreeMode	Sets the flag memory management when deleting items in the list
Total	Gets the number of elements in the list
IsSorted	Gets flag sorted list
SortMode	Gets the version of the sorting
Create methods	
CreateElement	Creates a new item to the list
Add methods	
Add	Adds element to the end of the list
Insert	Inserts element in the list in the specified position
Delete methods	
DetachCurrent	Remove an item from the current position of the list without deleting it "physically"
DeleteCurrent	Removes the element from the current position

	in the list
Delete	Removes the element from the specified position in the list
Clear	Removes all list items
Navigation	
IndexOf	Gets the index of the list item
GetNodeAtIndex	Gets an item with the specified index of the list
GetFirstNode	Gets the first element of the list
GetPrevNode	Gets the previous element list
GetCurrentNode	Gets the current list item
GetNextNode	Gets the next item in the list
GetLastNode	Gets the last item
Ordering methods	
Sort	Sort list
MoveToIndex	Moves the current item list to the specified position
Exchange	Changes elements of the list seats
Compare methods	
CompareList	Compares the list with another list
Search methods	
Search	Searches for an element equal to the model in sorted list
Input/output	
virtual Save	Saves data in the file list
virtual Load	Loads data from file list
virtual Type	Gets the type identifier list

FreeMode

Gets the flag memory management when deleting list items.

```
bool  FreeMode() const
```

Return Value

Flag of memory management.

Example:

```
//--- example for CList::FreeMode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get free mode flag
   bool list_free_mode=list.FreeMode();
   //--- delete list
   delete list;
  }
```

FreeMode

Sets the flag memory management when deleting list items.

```
void  FreeMode(
   bool  mode      // New value
   )
```

Parameters

mode

[in] New value of the flag memory management.

Note

Setting the flag memory management - an important point in the use of class CList.Since the elements of the list are pointers to dynamic objects, it is important to determine what to do with them when you delete from the list.If the flag is set, then when you delete an item from the list, the item is automatically deleted by the operator delete.If the flag is not set, it is assumed that a pointer to the deleted object is still somewhere in the user program and will be relieved of it (the program) then.

If the user resets the flag memory management, the user must understand their responsibility for the removal of items in the list before completing the program, because otherwise, is not freed memory occupied by the elements when they create new operator.When large amounts of data, it could lead, eventually, even to break your terminal.

If the user does not reset the flag memory management, there is another "reef".Using pointers-list items that are stored somewhere in the local variables, after removing the list, will lead to a critical error and crashes the program user.By default, the memory management flag is set, ie the class list, is responsible for freeing the memory elements.

Example:

```
//--- example for CList::FreeMode(bool)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- reset free mode flag
   list.FreeMode(false);
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
```

```
}
```

Total

Gets the number of elements in the list.

```
int  Total() const
```

Return Value

Number of elements in the list.

Example:

```
//--- example for CList::Total()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check total
   int total=list.Total();
   //--- use list
   //--- ...
   //--- delete list
   delete list;
  }
```

IsSorted

Gets the flag sorted list.

```
bool  IsSorted(
   int  mode=0      // Sorting mode
   ) const
```

Parameters

mode=0

[in] Tested version sorting

Return Value

Flag of the sorted list. If the list is sorted by the specified option? true, otherwise? false.

Note

Flag of the sorted list can not be changed directly. Flag set by Sort (int) and resets any methods to add / insert.

Example:

```
//--- example for CList::IsSorted()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check sorted
   if(list.IsSorted(0))
     {
      //--- use methods for sorted list
      //--- ...
     }
   //--- delete list
   delete list;
  }
```

SortMode

Gets the version of the sort.

```
int  SortMode() const
```

Return Value

Option sorting, or -1 if the list is not sorted.

Example:

```
//--- example for CList::SortMode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- check sort mode
   int sort_mode=list.SortMode();
   //--- use list
   //--- ...
   //--- delete list
   delete list;
  }
```

CreateElement

Creates a new item to the list.

```
CObject*  CreateElement()
```

Return Value

Pointer to the newly created element, if successful, NULL - if you can not create element.

Note

Method CreateElement () in class CList always returns NULL and does not perform any actions. If necessary, in a derived class, method CreateElement () should be implemented.

Example:

```
//--- example for CList::CreateElement(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   int    size=100;
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- fill list
   for(int i=0;i<size;i++)
     {
      CObject *object=list.CreateElement();
      if(object==NULL)
        {
         printf("Element create error");
         delete list;
         return;
        }
      list.Add(object);
     }
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

Add

Adds an element to the end of the list.

```
int  Add(
   CObject*  element      // Element to add
   )
```

Parameters

element

[in] Value of the element to add to the list.

Return Value

If successful, it returns the index of added element, or -1 in the case of error.

Note

Element is not added to the list, if the parameter does not pass valid pointer (ie NULL).

Example:

```
//--- example for CList::Add(Cobject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add 100 elements
   for(int i=0;i<100;i++)
     {
      if(list.Add(new CObject)==-1)
        {
         printf("Element addition error");
         delete list;
         return;
        }
     }
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

Insert

Inserts element in the list in the specified position.

```
int  Insert(
   CObject*  element,     // Element to insert
   int       pos          // Position
   )
```

Parameters

element

[in] value of the element to insert in the list

pos

[in] Position in the list to insert

Return Value

If successful, it returns the index of inserted element, or -1 in the case of error.

Note

Element is not added to the list, if the parameter does not pass valid pointer (ie NULL).

Example:

```
//--- example for CList::Insert(CObject*,int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- insert 100 elements
   for(int i=0;i<100;i++)
     {
      if(list.Insert(new CObject,0)==-1)
        {
         printf("Element insert error");
         delete list;
         return;
        }
     }
   //--- use list
   //--- . . .
   //--- delete list
```

```
   delete list;
  }
```

DetachCurrent

Extracts an element from the current position without its "physical" deleteion.

```
CObject*  DetachCurrent()
```

Return Value

Pointer to the removal of elements in case of success, NULL - if you can not remove the element.

Note

When removed from the list, the item is not removed in any state of the flag memory management. Pointer to withdraw from the list of ingredients of the release element after use.

Example:

```
//--- example for CList::DetachCurrent()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.DetachCurrent();
   if(object==NULL)
     {
      printf("Detach error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- delete element
   delete object;
   //--- delete list
   delete list;
  }
```

DeleteCurrent

Removes the element from the current position in the list.

```
bool  DeleteCurrent()
```

Return Value

true if successful, false - if you can not remove the element.

Note

If enabled memory management, memory, removes the element is released.

Example:

```
//--- example for CList::DeleteCurrent()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   if(!list.DeleteCurrent())
     {
      printf("Delete error");
      delete list;
      return;
     }
   //--- delete list
   delete list;
  }
```

Delete

Removes the element from the given position in the list.

```
bool  Delete(
   int  pos      // Position
   )
```

Parameters

pos

[in] position removes the element in the list.

Return Value

true if successful, false - if you can not remove the element.

Note

If enabled memory management, memory, removes the element is released.

Example:

```
//--- example for CList::Delete(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   if(!list.Delete(0))
     {
      printf("Delete error");
      delete list;
      return;
     }
   //--- delete list
   delete list;
  }
```

Clear

Removes all elements of the list.

```
void  Clear()
```

Note

If enabled memory management, memory, deleted items are exempt.

Example:

```
//--- example for CList::Clear()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   //--- clear list
   list.Clear();
   //--- delete list
   delete list;
  }
```

IndexOf

Gets the index of the list item.

```
int  IndexOf(
   CObject*  element      // Pointer to the element
   )
```

Parameters

element

[in] Pointer to the list item.

Return Value

Index item in the list, or -1.

Example:

```
//--- example for CList::IndexOf(CObject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   CObject *object=new CObject;
   if(object==NULL)
     {
      printf("Element create error");
      delete list;
      return;
     }
   if(list.Add(object))
     {
      int pos=list.IndexOf(object);
     }
   //--- delete list
   delete list;
  }
```

GetNodeAtIndex

Gets an item with the specified index of the list.

```
CObject*  GetNodeAtIndex(
   int  pos      // position
   )
```

Parameters

pos

[in] item index in the list.

Returned value

Pointer to the item in case of success, NULL - if you can not get a pointer.

Example:

```
//--- example for CList::GetNodeAtIndex(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetNodeAtIndex(10);
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetFirstNode

Gets the first element of the list.

```
CObject*  GetFirstNode()
```

Return Value

Pointer to the first item in case of success, NULL - if you can not get a pointer.

Example:

```
//--- example for CList::GetFirstNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetFirstNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetPrevNode

Gets the previous element of the list.

```
CObject*  GetPrevNode()
```

Return Value

Pointer to the previous element, if successful, NULL - if you can not get a pointer.

Example:

```
//--- example for CList::GetPrevNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetPrevNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetCurrentNode

Gets the current list item.

```
CObject*  GetCurrentNode()
```

Return Value

Pointer to the current item, if successful, NULL - if you can not get a pointer.

Example:

```
//--- example for CList::GetCurrentNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetCurrentNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetNextNode

Gets the next item in the list.

```
CObject*  GetNextNode()
```

Return Value

Pointer to the next item if successful, NULL - if you can not get a pointer.

Example:

```
//--- example for CList::GetNextNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetNextNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

GetLastNode

Gets the last element of the list.

```
CObject*  GetLastNode()
```

Return Value

Pointer to the last element in the case of success, NULL - if you can not get a pointer.

Example:

```
//--- example for CList::GetLastNode()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add list elements
   //--- . . .
   CObject *object=list.GetLastNode();
   if(object==NULL)
     {
      printf("Get node error");
      delete list;
      return;
     }
   //--- use element
   //--- . . .
   //--- do not delete element
   //--- delete list
   delete list;
  }
```

Sort

Sorts a list.

```
void  Sort(
   int  mode      // Sorting mode
   )
```

Parameters

mode

[in] Sorting mode.

Return Value

No.

Note

Sorting the list is always in ascending order.

Example:

```
//--- example for CList::Sort(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- sorting by mode 0
   list.Sort(0);
   //--- use list
   //--- ...
   //--- delete list
   delete list;
  }
```

MoveToIndex

Moves the current item list to the specified position.

```
bool  MoveToIndex(
   int  pos      // Position
   )
```

Parameters

pos

[in] Position in the list to move.

Return Value

true if successful, false - if you can not move the item.

Example:

```
//--- example for CList::MoveToIndex(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- move current node to begin
   list.MoveToIndex(0);
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

Exchange

Changes elements of the list seats.

```
bool  Exchange(
   CObject*  node1,      // List item
   CObject*  node2       // List item
   )
```

Parameters

node1

[in] List item

node2

[in] List item

Return Value

true if successful, false - if you can not change the elements in some places.

Example:

```
//--- example for CList::Exchange(CObject*,CObject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- exchange
   list.Exchange(list.GetFirstNode(),list.GetLastNode());
   //--- use list
   //--- . . .
   //--- delete list
   delete list;
  }
```

CompareList

Compares the list with another list.

```
bool  CompareList(
   CList*  list      // With whom we compare
   )
```

Parameters

list

[in] A pointer to an instance of class CList-source elements for comparison.

Return Value

true if the lists are equal, false - if not.

Example:

```
//--- example for CList::CompareList(const CList*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- create source list
   CList *src=new CList;
   if(src==NULL)
     {
      printf("Object create error");
      delete list;
      return;
     }
   //--- fill source list
   //--- . . .
   //--- compare with another list
   bool result=list.CompareList(src);
   //--- delete lists
   delete src;
   delete list;
  }
```

Search

Searches for an element equal to the model in the sorted list.

```
CObject*  Search(
   CObject*  element      // Sample
   )
```

Parameters

element

[in] Sample cell to search for in the list.

Return Value

Pointer to the found item if successful, NULL - if the item was not found.

Example:

```
//--- example for CList::Search(CObject*)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add lists elements
   //--- . . .
   //--- sort list
   list.Sort(0);
   //--- create sample
   CObject *sample=new CObject;
   if(sample==NULL)
     {
      printf("Sample create error");
      delete list;
      return;
     }
   //--- set sample attributes
   //--- . . .
   //--- search element
   if(list.Search(sample)!=NULL) printf("Element found");
   else                          printf("Element not found");
   //--- delete list
   delete list;
  }
```

Save

Saves data in the file list.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the previously opened using the function FileOpen (...) file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CList::Save(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CList *list=new CList;
   //---
   if(list!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- add lists elements
   //--- . . .
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!list.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         delete list;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- delete list
   delete list;
```

```
}
```

Load

Loads list data from a file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] Handle of the previously open, with the function FileOpen (...), binary

Return Value

true - if successfully completed, false - if an error.

Note

When reading from the file list items to create each element of the method is called CList:: CreateElement ().

Example:

```
//--- example for CLoad::Load(int)
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   int    file_handle;
   CList *list=new CList;
   //---
   if(list!=NULL)
     {
      printf("Object create error");
      return;
     }
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!list.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         delete list;
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
```

```
   //--- use list elements
   //--- . . .
   //--- delete list
   delete list;
  }
```

Type

Gets the type identifier list.

```
virtual int  Type()
```

Return Value

Type identifier list (for CList - 7779).

Example:

```
//--- example for CList::Type()
#include <Arrays\List.mqh>
//---
void OnStart()
  {
   CList *list=new CList;
   //---
   if(list==NULL)
     {
      printf("Object create error");
      return;
     }
   //--- get list type
   int type=list.Type();
   //--- delete list
   delete list;
  }
```

CTreeNode

Class CTreeNode is a class of node of the binary tree CTree.

Description

Class CTreeNode provides the ability to work with nodes of the binary tree CTree. Options of navigation through the tree is implemented in the class. Besides that methods of work with a file are implemented.

Declaration

```
class CTreeNode : public CObject
```

Title

```
#include <Arrays\TreeNode.mqh>
```

Class Methods

Attributes	
Owner	Gets/sets the pointer of the owner node
Left	Gets/sets the pointer of the left node
Right	Gets/sets the pointer of the right node
Balance	Gets the node balance
BalanceL	Gets the balance of the left sub-branch of the node
BalanceR	Gets the balance of the right sub-branch of the node
Creation of a new element	
CreateSample	Creates a new node instance
Comparison	
RefreshBalance	Recalculates the node balance
Search	
GetNext	Gets the pointer of the next node
Input/Output	
SaveNode	Saves the node data to a file
LoadNode	Downloads the node data from a file
virtual Type	Gets the identifier of the node type

Derived classes:

- CTree

Trees of CTreeNode class descendants get practical application.

A descendant of class CTreeNode must have predefined methods: CreateSample that creates a new instance of the descendant class of CTreeNode, Compare that compares values of key fields of the descendant class of CTreeNode, Type (if it's necessary to identify a node), SaveNode and LoadNode (if it's necessary to work with a file).

Let's consider and example of a CTree descendant class.

```
//+------------------------------------------------------------------+
//|                                                   MyTreeNode.mq5 |
//|                        Copyright 2010, MetaQuotes Software Corp. |
//|                                        http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
//---
#include <Arrays\TreeNode.mqh>
//+------------------------------------------------------------------+
//| Describe classderived from CTreeNode.                            |
//+------------------------------------------------------------------+
//| Class CMyTreeNode.                                               |
//| Purpose: Class of element of a binary tree.                      |
//|              Descendant of class CTreeNode.                      |
//+------------------------------------------------------------------+
class CMyTreeNode : public CTreeNode
  {
protected:
   //--- user's data
   long              m_long;           // key field of type long
   double            m_double;         // custom variable of type double
   string            m_string;         // custom variable of type string
   datetime          m_datetime;       // custom variable of type datetime

public:
                     CMyTreeNode();
   //--- methods of accessing these user's data
   long              GetLong(void)                 { return(m_long); }
   void              SetLong(long value)           { m_long=value;  }
   double            GetDouble(void)               { return(m_double); }
   void              SetDouble(double value)       { m_double=value;  }
   string            GetString(void)               { return(m_string); }
   void              SetString(string value)       { m_string=value;  }
   datetime          GetDateTime(void)             { return(m_datetime); }
   void              SetDateTime(datetime value)   { m_datetime=value;  }
   //--- methods of working with files
   virtual bool      Save(int file_handle);
   virtual bool      Load(int file_handle);
```

```
protected:
   virtual int       Compare(const CObject *node,int mode);
   //--- methods of creating class instances
   virtual CTreeNode* CreateSample();
  };
//+------------------------------------------------------------------+
//| CMyTreeNode class constructor.                                   |
//| INPUT:  none.                                                    |
//| OUTPUT: none.                                                    |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
void CMyTreeNode::CMyTreeNode()
  {
//--- initialization of user's data
   m_long        =0;
   m_double      =0.0;
   m_string      ="";
   m_datetime    =0;
  }
//+------------------------------------------------------------------+
//| Comparison with another three node by the specified algorithm.   |
//| INPUT:  node - array element to compare,                         |
//|         mode - identifier of comparison algorithm.               |
//| OUTPUT: result of comparison (>0,0,<0).                          |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
int CMyTreeNode::Compare(const CObject *node,int mode)
  {
//--- parameter mode is ignored, because tree construction algorithm is the only one
   int res=0;
//--- explicit type casting
   CMyTreeNode *n=node;
   res=(int)(m_long-n.m_long);
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| Creation of a new class instance.                                |
//| INPUT:  none.                                                    |
//| OUTPUT: pointer to a new instance of class CMyTreeNode.          |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
CTreeNode* CMyTreeNode::CreateSample()
  {
   CMyTreeNode *result=new CMyTreeNode;
//---
   return(result);
  }
```

```
//+------------------------------------------------------------------+
//| Write tree node data to a file.                                  |
//| INPUT:  file_handle -handle of a file pre-opened for writing.    |
//| OUTPUT: true if OK, otherwise false.                             |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
bool CMyTreeNode::Save(int file_handle)
  {
   uint i=0,len;
//--- checks
   if(file_handle<0) return(false);
//--- writing user data
//--- writing custom variable of type long
   if(FileWriteLong(file_handle,m_long)!=sizeof(long))          return(false);
//--- writing custom variable of type double
   if(FileWriteDouble(file_handle,m_double)!=sizeof(double))    return(false);
//--- writing custom variable of type string
   len=StringLen(m_string);
//--- write string length
   if(FileWriteInteger(file_handle,len,INT_VALUE)!=INT_VALUE)   return(false);
//--- write the string
   if(len!=0 && FileWriteString(file_handle,m_string,len)!=len) return(false);
//--- writing custom variable of type datetime
   if(FileWriteLong(file_handle,m_datetime)!=sizeof(long))      return(false);
//---
   return(true);
  }
//+------------------------------------------------------------------+
//| Read tree node data from a file.                                 |
//| INPUT:  file_handle -handle of a file pre-opened for reading.    |
//| OUTPUT: true if OK, otherwise false.                             |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
bool CMyTreeNode::Load(int file_handle)
  {
   uint i=0,len;
//--- checks
   if(file_handle<0) return(false);
//--- reading
   if(FileIsEnding(file_handle)) return(false);
//--- reading custom variable of type char
//--- reading custom variable of type long
   m_long=FileReadLong(file_handle);
//--- reading custom variable of type double
   m_double=FileReadDouble(file_handle);
//--- reading custom variable of type string
//--- read the string length
   len=FileReadInteger(file_handle,INT_VALUE);
```

```
//--- read the string
   if(len!=0) m_string=FileReadString(file_handle,len);
   else       m_string="";
//--- reading custom variable of type datetime
   m_datetime=FileReadLong(file_handle);
//---
   return(true);
  }
```

Owner

Gets the pointer of the owner node.

```
CTreeNode*  Owner()
```

Return Value

Pointer of the node-owner.

Owner

Sets the pointer of the owner node.

```
void  Owner(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] New value of the pointer of the owner node.

Return Value

None.

Left

Gets the pointer of the left node.

```
CTreeNode*  Left()
```

Return Value

Pointer of the left node.

Left

Sets the pointer of the left node.

```
void  Left(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] New value of the pointer of the left node.

Return Value

None.

Right

Gets the pointer of the right node.

```
CTreeNode*  Right()
```

Return Value

The pointer of the right node.

Right

Sets the pointer of the right node.

```
void  Right(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] New value of the pointer of the right node.

Return Value

None.

Balance

Gets the node balance.

```
int  Balance() const
```

Return Value

Node balance.

BalanceL

Gets the balance of the left sub-branch of the node.

```
int  BalanceL() const
```

Return Value

Balance of the left sub-branch of the node.

BalanceR

Gets the balance of the right sub-branch of the node.

```
int  BalanceR() const
```

Return Value

Balance of the right sub-branch of the node.

CreateSample

Creates a new node sample.

```
virtual CTreeNode*  CreateSample()
```

Return Value

Pointer to the new node sample or NULL.

RefreshBalance

Recalculates the node balance.

```
int  RefreshBalance()
```

Return Value

Node balance.

GetNext

Gets the pointer of the next node.

```
CTreeNode*  GetNext(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] Node of the search start.

Return Value

Pointer of the next node.

SaveNode

Writes node data to a file.

```
bool  SaveNode(
   int  file_handle      // handle
   )
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for writing.

Return Value

true in case of success, otherwise false.

LoadNode

Reads node data from a file.

```
bool  LoadNode(
   int          file_handle,      // handle
   CTreeNode*   main              // node
   )
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for reading.

main

[in] Node for data.

Return Value

true in case of success, otherwise false.

Type

Gets the identifier of the node type.

```
virtual int  Type() const
```

Return Value

Identifier of the node type.

CTree

Class CTree is a class of the binary tree of samples of class CTreeNode and its descendants.

Description

Class CTree privides the possibility to work with a binary tree of CTreeNode class samples and its descendants. Options of adding/inserting/deleting of three elements and search in a tree are implemented in the class. Besides that, methods of work with a file are implemented.

Note that mechanism of dynamic memory management is not implemented in class CTree (as distinct from classes CList and CArrayObj). All tree nodes are deleted with memory release.

Declaration

```
class CTree : public CTreeNode
```

Title

```
#include <Arrays\Tree.mqh>
```

Class Methods

Attributes	
Root	Gets a root node of the tree
Creation of a new element	
CreateElement	Creates a new node instance
Filling	
Insert	Adds a node to a tree
Deletion	
Detach	Detaches a specified node from a tree
Delete	Deletes a specified node from a tree
Clear	Deletes all nodes of a tree
Search	
Find	Searches for a node in a tree by sample
Input/output	
virtual Save	Saves all the tree data to a file
virtual Load	Downloads tree data from a file
virtual Type	Gets identifier of the tree type

Trees of CTreeNode class descendants - descendants of class CTree get practical application.

Descendant of class CTree must have a predefined method CreateElement that creates a new sample of descendant class CTreeNode.

Let's consider an example of descendant class CTree.

```
//+------------------------------------------------------------------+
//|                                                      MyTree.mq5 |
//|                        Copyright 2010, MetaQuotes Software Corp. |
//|                                              http://www.metaquotes.net/ |
//+------------------------------------------------------------------+
#property copyright "2010, MetaQuotes Software Corp."
#property link      "http://www.mql5.com"
//---
#include <Arrays\Tree.mqh>
#include "MyTreeNode.mqh"
//---
input int extCountedNodes = 100;
//+------------------------------------------------------------------+
//| Describe class CMyTree derived from CTree.                       |
//+------------------------------------------------------------------+
//| Class CMyTree.                                                   |
//| Purpose: Construction and navigation of a binary search tree.    |
//+------------------------------------------------------------------+
class CMyTree : public CTree
  {
public:
   //--- methods of search on the tree by custom data
   CMyTreeNode*         FindByLong(long find_long);
   //--- method of creation of the tree element
   virtual CTreeNode *CreateElement();
  };
//---
CMyTree MyTree;
//+------------------------------------------------------------------+
//| Creation of a new tree node.                                     |
//| INPUT:  none.                                                    |
//| OUTPUT: pointer to the new tree node of OK, or NULL.             |
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
CTreeNode *CMyTree::CreateElement()
  {
   CMyTreeNode *node=new CMyTreeNode;
//---
   return(node);
  }
//+------------------------------------------------------------------+
//| Search of element in a list by value m_long.                     |
//| INPUT:  find_long - searched value.                              |
//| OUTPUT: pointer of a found list element, or NULL.                |
```

```
//| REMARK: none.                                                    |
//+------------------------------------------------------------------+
CMyTreeNode* CMyTree::FindByLong(long find_long)
  {
   CMyTreeNode *res=NULL;
   CMyTreeNode *node;
//--- create a tree node to pass the search parameter
   node=new CMyTreeNode;
   if(node==NULL) return(NULL);
   node.SetLong(find_long);
//---
   res=Find(node);
   delete node;
//---
   return(res);
  }
//+------------------------------------------------------------------+
//| script "testing of class CMyTree"                                |
//+------------------------------------------------------------------+
//---  array for string initialization
string str_array[11]={"p","oo","iii","uuuu","yyyyy","ttttt","rrrr","eee","ww","q","99
//---
int OnStart() export
  {
   int          i;
   uint         pos;
   int          beg_time,end_time;
   CMyTreeNode *node; //--- temporary pointer to the sample of class CMyTreeNode
//---
   printf("Start test %s.",__FILE__);
//--- Fill out MyTree with samples of class MyTreeNode in the amount of extCountedNod
   beg_time=GetTickCount();
   for(i=0;i<extCountedNodes;i++)
     {
      node=MyTree.CreateElement();
      if(node==NULL)
        {
         //--- emergency exit
         printf("%s (%4d): create error",__FILE__,__LINE__);
         return(__LINE__);
        }
      NodeSetData(node,i);
      node.SetLong(i);
      MyTree.Insert(node);
     }
   end_time=GetTickCount();
   printf("Filling time of MyTree is %d ms.",end_time-beg_time);
//--- Create a temporary tree TmpMyTree.
```

```
   CMyTree TmpMyTree;
//--- Detach 50% of tree elements (all even)
//--- and add them to the temporary tree TmpMyTree.
   beg_time=GetTickCount();
   for(i=0;i<extCountedNodes;i+=2)
     {
      node=MyTree.FindByLong(i);
      if(node!=NULL)
         if(MyTree.Detach(node)) TmpMyTree.Insert(node);
     }
   end_time=GetTickCount();
   printf("Deletion time of %d elements from MyTree is %d ms.",extCountedNodes/2,end_
//--- Return the detached
   node=TmpMyTree.Root();
   while(node!=NULL)
     {
      if(TmpMyTree.Detach(node)) MyTree.Insert(node);
      node=TmpMyTree.Root();
     }
//--- Check work of method Save(int file_handle);
   int file_handle;
   file_handle=FileOpen("MyTree.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!MyTree.Save(file_handle))
        {
         //--- error writing to a file
         //--- emergency exit
         printf("%s: Error %d in %d!",__FILE__,GetLastError(),__LINE__);
         //--- close file before leaving!!!
         FileClose(file_handle);
         return(__LINE__);
        }
      FileClose(file_handle);
     }
//--- Check work of method Load(int file_handle);
   file_handle=FileOpen("MyTree.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!TmpMyTree.Load(file_handle))
        {
         //--- error reading from file
         //--- emergency exit
         printf("%s: Error %d in %d!",__FILE__,GetLastError(),__LINE__);
         //--- close file before leaving!!!
         FileClose(file_handle);
         return(__LINE__);
        }
```

```
      FileClose(file_handle);
     }
//---
   MyTree.Clear();
   TmpMyTree.Clear();
//---
   printf("End test %s. OK!",__FILE__);
//---
   return(0);
  }
//+------------------------------------------------------------------+
//| Function of output of node contents to journal                   |
//+------------------------------------------------------------------+
void NodeToLog(CMyTreeNode *node)
  {
   printf("   %I64d,%f,'%s','%s'",
              node.GetLong(),node.GetDouble(),
              node.GetString(),TimeToString(node.GetDateTime()));
  }
//+------------------------------------------------------------------+
//| Function of "filling" of node with random values                 |
//+------------------------------------------------------------------+
void NodeSetData(CMyTreeNode *node,int mode)
  {
   if(mode%2==0)
     {
      node.SetLong(mode*MathRand());
      node.SetDouble(MathPow(2.02,mode)*MathRand());
     }
   else
     {
      node.SetLong(mode*(long)(-1)*MathRand());
      node.SetDouble(-MathPow(2.02,mode)*MathRand());
     }
   node.SetString(str_array[mode%10]);
   node.SetDateTime(10000*mode);
  }
```

Root

Gets the root node of the tree.

```
CTreeNode*  Root() const
```

Return Value

Pointer of the root node of the tree.

CreateElement

Creates a new instance of the node.

```
virtual CTreeNode*  CreateElement()
```

Return Value

Pointer of the new instance of the node or NULL.

Insert

Adds a node to a tree.

```
CTreeNode*  Insert(
   CTreeNode*  new_node      // node
   )
```

Parameters

new_node

[in] pointer of a node to insert to a tree.

Return Value

Pointer of the owner node or NULL.

Detach

Detaches a specified node from a tree.

```
bool  Detach(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] Node pointer to detach.

Return Value

true in case of success, otherwise false.

Note

After detachment the node pointer is not released. The tree is balanced.

Delete

Deletes a specified node from a tree.

```
bool  Delete(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] Node pointer to delete.

Return Value

true in case of success, otherwise false.

Note

After deletion a node pointer is released. The tree is balanced.

Clear

Deletes all nodes of a tree.

```
void  Clear()
```

Return Value

None.

Note

After deletion node pointers are released.

Find

Searches for a node in a tree by sample.

```
CTreeNode*  Find(
   CTreeNode*  node      // node
   )
```

Parameters

node

[in] Node that contains data-search sample.

Return Value

Pointer of the found node or NULL.

Save

Writes tree data to a file.

```
virtual bool  Save(
   int  file_handle      // handle
   )
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for writing.

Return Value

true in case of success, otherwise false.

Load

Reads tree data to a file.

```
virtual bool  Load(
   int  file_handle      // handle
   )
```

Parameters

file_handle

[in] Handle of a binary file that was earlier opened for reading.

Return Value

true in case of success, otherwise false.

Type

Gets identifier of the tree type.

```
virtual int  Type() const
```

Return Value

Identifier of the tree type.

Graphic Objects

This section contains the technical details of working with classes of graphical objects and a description of the relevant components of the standard library MQL5.

The use of classes of graphical objects, will save time when creating custom programs (scripts, expert).

Standard library MQL5 (in terms of graphical objects) is placed in the working directory of the terminal in the folder Include \ ChartObjects.

Class/Group	Description
Base class for graphical object CChartObject	Base class of a graphic object
Lines	Group classes "Lines"
Channels	Group classes "Channels"
Gann Tools	Group classes "Gann"
Fibonacci Tools	Group classes "Fibonacci"
Elliott Tools	Group classes "Elliott"
Shapes	Group classes "Shapes"
Arrows	Group classes "Arrows"
Controls	Group classes "Controls"

CChartObject

CChartObject is a base class of graphic objects of chart type of the Standard MQL5 library.

Description

Class CChartObject provides the simplified access for all of its descendants to MQL5 API functions.

Declaration

```
class CChartObject : public CObject
```

Title

```
#include <ChartObjects\ChartObject.mqh>
```

Class Methods

Attributes	
ChartId	Gets the ID chart, who owns a graphic
Window	Gets the number of windows in which the chart is a graphic
Name	Gets/sets the name of a graphic object
NumPoints	Gets the number of anchor points
Assign	
Attach	Binds a graphic chart
SetPoint	Sets the anchor point
Delete	
Delete	Deletes a graphic chart
Detach	Untie a graphic chart
Shift	
ShiftObject	The relative movement of the object
ShiftPoint	The relative movement of the object point
Object properties	
Time	Gets/sets the time coordinates of the object point
Price	Gets/sets the price coordinate of a point object
Color	Gets/sets the color of the object
Style	Gets/sets the line style object
Width	Gets/sets the width of the line object

BackGround	Gets/sets the flag drawing object background
Selected	Gets/sets the flag sit on an object
Selectable	Gets/sets the flag selectable object
Description	Gets/sets the text of the object
Tooltip	Gets/sets the tooltip of the object
Timeframes	Gets/sets the mask of flags visibility of the object
Z_Order	Gets/sets the priority of clicking on a chart
CreateTime	Gets the time object creation
Levels properties of the object	
LevelsCount	Gets/sets the number of levels of object
LevelColor	Gets/sets the color of the line level
LevelStyle	Gets/sets the line style level
LevelWidth	Gets/sets the width of the line level
LevelValue	Gets/sets the level
LevelDescription	Gets/sets the text level
Access to MQL5 API functions	
GetInteger	Gets the value of the object properties
SetInteger	Sets the object properties
GetDouble	Gets the value of the object properties
SetDouble	Sets the object properties
GetString	Gets the value of the object properties
SetString	Sets the object properties
Input/Output	
virtual Save	Virtual method entry in the file
virtual Load	Virtual method of reading from a file
virtual Type	Virtual method of identification

Derived classes:

- CChartObjectArrow
- CChartObjectBitmap
- CChartObjectBmpLabel
- CChartObjectCycles
- CChartObjectElliottWave3

- CChartObjectEllipse
- CChartObjectFiboArc
- CChartObjectFiboFan
- CChartObjectFiboTimes
- CChartObjectHLine
- CChartObjectRectangle
- CChartObjectSubChart
- CChartObjectText
- CChartObjectTrend
- CChartObjectTriangle
- CChartObjectVLine

ChartId

Gets the ID chart, who owns a graphic object.

```
long  ChartId() const
```

Return Value

Id chart on which the graphic object. If object not found, it returns -1.

Example:

```
//--- example for CChartObject::ChartId
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get chart idintifier of chart object
   long chatr_id=object.ChartId();
  }
```

Window

Gets the number of windows in which the chart is a graphic object.

```
int  Window() const
```

Return Value

Number of windows in which the chart is a graphic object (0 - main window). If object not found, it returns -1.

Example:

```
//--- example for CChartObject::Window
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get window of chart object
   int window=object.Window();
  }
```

Name (Get Method)

Gets the name of the graphic object.

```
string  Name() const
```

Return Value

Name of the graphic object tied to an instance of the class. If object not found, returns NULL.

Name (Set Method)

Sets the name of the graphic object.

```
bool  Name(
   string  name      // new name
   )
```

Parameters

name

[in] The new name of the graphic object.

Return Value

true if successful, false - if you can not change the name.

Example:

```
//--- example for CChartObject::Name
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get name of chart object
   string object_name=object.Name();
   if(object_name!="MyChartObject")
     {
     //--- set name of chart object
     object.Name("MyChartObject");
     }
  }
```

NumPoints

Gets the number of anchor points of a graphic object.

```
int  NumPoints() const
```

Return Value

Number of points linking a graphic object that is bound to an instance of the class. If not bound object, it returns 0.

Example:

```
//--- example for CChartObject::NumPoints
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get points count of chart object
   int points=object.NumPoints();
  }
```

Attach

Binds a graphical object to an instance of the class.

```
bool  Attach(
   long     chart_id,      // Chart ID
   string   name,          // Name of the object
   int      window,        // Chart window
   int      points         // Number of points
   )
```

Parameters

chart_id

[out] Chart identifier.

name

[in] Name of the graphic object.

window

[in] Chart window number (0 - main window).

points

[in] Number of points anchor graphic object.

Return Value

true - if successful, false - if you can not bind object.

Example:

```
//--- example for CChartObject::Attach
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- attach chart object
   if(!object.Attach(ChartID(),"MyObject",0,2))
     {
      printf("Object attach error");
      return;
     }
  }
```

SetPoint

Sets the new coordinates of this anchor point graphic object.

```
bool  SetPoint(
   int       point,          // Point number
   datetime  new_time,       // Time coordinate
   double    new_price       // Price coordinate
   )
```

Parameters

point

[in] Number anchor point graphic object.

new_time

[in] The new value of the coordinates of this point of time bindings.

new_price

[in] New value coordinates of the price specified anchor point.

Return Value

true - if successful, false - if you can not change the coordinates of the point.

Example:

```
//--- example for CChartObject::SetPoint
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   double       price;
   //---
   if(object.NumPoints()>0)
     {
      //--- set point of chart object
      object.SetPoint(0,CurrTime(),price);
     }
  }
```

Delete

Removes a graphical object with the attached chart.

```
bool  Delete()
```

Return Value

true - if successful, false - if you can not remove the object.

Example:

```
//--- example for CChartObject::Delete
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- detach chart object
   if(!object.Delete())
     {
      printf("Object delete error");
      return;
     }
  }
```

Detach

Untie graphic object.

```
bool  Detach()
```

Return Value

true - if successful, false - if you can not decouple the object.

Example:

```
//--- example for CChartObject::Detach
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- detach chart object
   if(!object.Detach())
     {
      printf("Object detach error");
      return;
     }
  }
```

ShiftObject

Moves a graphic object.

```
bool  ShiftObject(
   datetime  d_time,      // Increment of time coordinate
   double    d_price      // Increment of price coordinate
   )
```

Parameters

d_time

[in] Increment the coordinates of all points of time bindings

d_price

[in] Increment the coordinates of the prices of all waypoints.

Return Value

true - if successful, false - if you can not move the object.

Example:

```
//--- example for CChartObject::ShiftObject
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   datetime     d_time;
   double       d_price;
   //--- shift chart object
   object.ShiftObject(d_time,d_price);
  }
```

ShiftPoint

Moves a specified point anchor graphic.

```
bool  ShiftPoint(
   int       point,       // Point number
   datetime  d_time,      // Increment of time coordinate
   double    d_price      // Increment of price coordinate
   )
```

Parameters

point

[in] Number anchor point graphic object.

d_time

[in] Increment the coordinates of time specified anchor point.

d_price

[in] Increment the coordinates of the price specified anchor point.

Return Value

true - if successful, false - if you can not move the point.

Example:

```
//--- example for CChartObject::ShiftPoint
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   datetime     d_time;
   double       d_price;
   //---
   if(object.NumPoints()>0)
     {
      //--- shift point of chart object
      object.ShiftPoint(0,d_time,d_price);
     }
  }
```

Time (Get Method)

The coordinates of time specified anchor point graphic object.

```
datetime  Time(
   int  point      // Point number
   ) const
```

Parameters

point

[in] Number anchor point graphic object.

Return Value

Coordinates of time specified anchor point graphic object that is bound to an instance of the class. If not bound object or the object is not this point, it returns 0.

Time (Set Method)

Sets the coordinate of time specified anchor point graphic object.

```
bool  Time(
   int       point,         // Point number
   datetime  new_time       // Time
   )
```

Parameters

point

[in] Number anchor point graphic object.

new_time

[in] The new value of the coordinates of this point of time anchor graphic object.

Return Value

true - if successful, false - if you can not change the coordinate of time.

Example:

```
//--- example for CChartObject::Time
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.NumPoints();i++)
     {
      //--- get time of point chart object
      datetime point_time=object.Time(i);
      if(point_time==0)
```

```
         {
          //--- set time of point chart object
          object.Time(i,TimeCurrent());
         }
      }
   }
```

Price (Get Method)

Gets the coordinate of the price specified anchor point graphic object.

```
double  Price(
   int  point      // Point number
   ) const
```

Parameters

point

[in] Number anchor point graphic object.

Return Value

Coordinate prices specified anchor point graphic object that is bound to an instance of the class. If not bound object or the object is not this point, it returns EMPTY_VALUE.

Price (Set Method)

Sets the coordinate of the price specified anchor point graphic object.

```
bool  Price(
   int     point,          // Point number
   double  new_price       // Price
   )
```

Parameters

point

[in] Number anchor point graphic object.

new_price

[in] News value coordinates of the price specified anchor point graphic object.

Return Value

true - if successful, false - if you can not change the coordinate prices.

Example:

```
//--- example for CChartObject::Price
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   double       price;
   //---
   for(int i=0;i<object.NumPoints();i++)
     {
      //--- get price of point chart object
      double point_price=object.Price(i);
```

```
      if(point_price!=price)
        {
         //--- set price of point chart object
         object.Price(i,price);
        }
     }
  }
```

Color (Get Method)

Gets the line color of the graphic object.

```
color  Color() const
```

Return Value

Color line of graphic object, assigned to the class instance. If there is no object assigned, it returns CLR_NONE.

Color (Set Method)

Sets the color of the line for the graphic object.

```
bool  Color(
   color  new_color      // New color
   )
```

Parameters

new_color

[in] New value line color graphic object.

Return Value

true - if successful, false - if you can not change the color.

Example:

```
//--- example for CChartObject::Color
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get color of chart object
   color object_color=object.Color();
   if(object_color!=clrRed)
     {
     //--- set color of chart object
     object.Color(clrRed);
     }
  }
```

Style (Get Method)

Gets the line style graphic.

```
ENUM_LINE_STYLE  Style() const
```

Return Value

Style line of the graphic object, assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Style (Set Method)

Sets the line style graphic.

```
bool  Style(
   ENUM_LINE_STYLE  new_style      // Style
   )
```

Parameters

new_style

[in] New value-style line drawing object.

Return Value

true - if successful, false - if you can not change the style.

Example:

```
//--- example for CChartObject::Style
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get style of chart object
   ENUM_LINE_STYLE style=object.Style();
   if(style!=STYLE_SOLID)
     {
      //--- set style of chart object
      object.Style(STYLE_SOLID);
     }
  }
```

Width (Get Method)

Gets the thickness of the line graphic object.

```
int  Width() const
```

Return Value

The thickness of the line graphic object that is bound to an instance of the class. If not bound object, it returns -1.

Width (Set Method)

Sets the thickness of the line graphic object.

```
bool  Width(
   int  new_width      // Thickness
   )
```

Parameters

new_width

[in] The new value of the thickness of the line graphic object.

Return Value

true - if successful, false - if you can not change the thickness.

Example:

```
//--- example for CChartObject::Width
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get width of chart object
   int width=object.Width();
   if(width!=1)
     {
      //--- set width of chart object
      object.Width(1);
     }
  }
```

Background (Get Method)

Gets the flag drawing a graphic object in the background.

```
bool  Background() const
```

Return Value

Flag of drawing in the background, a graphic object that is bound to an instance of the class. If not bound object returns false.

Background (Set Method)

Sets the flag drawing a graphic object in the background.

```
bool  Background(
   bool  background      // Value of the flag
   )
```

Parameters

background

[in] New value of the flag drawing a graphic object in the background.

Return Value

true - if successful, false - if you can not change the flag.

Example:

```
//--- example for CChartObject::Background
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get background flag of chart object
   bool background_flag=object.Background();
   if(!background_flag)
     {
     //--- set background flag of chart object
     object.Background(true);
     }
  }
```

Selected (Get Method)

Gets the flag "reprimand" graphic object.

```
bool  Selected() const
```

Return Value

Flag "slate", a graphic object that is bound to an instance of the class. If not bound object returns false.

Selected (Set Method)

Sets the flag "reprimand" graphic object.

```
bool  Selected(
   bool  selected      // Value of the flag
   )
```

Parameters

selected

[in] New value of the flag "reprimand" graphic object.

Return Value

true - if successful, false - if you can not change the flag.

Example:

```
//--- example for CChartObject::Selected
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get selected flag of chart object
   bool selected_flag=object.Selected();
   if(selected_flag)
     {
     //--- set selected flag of chart object
     object.Selected(false);
     }
  }
```

Selectable (Get Method)

Gets the flag "selectable" graphic object.

```
bool  Selectable() const
```

Return Value

Flag "selectable", a graphic object that is bound to an instance of the class. If not bound object returns false.

Selectable (Set Method)

Sets the flag "selectable" graphic object.

```
bool  Selectable(
   bool  selectable      // Value of the flag
   )
```

Parameters

selectable

[in] New value of the flag "selectable" graphic object.

Return Value

true - if successful, false - if you can not change the flag.

Example:

```
//--- example for CChartObject::Selectable
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get selectable flag of chart object
   bool selectable_flag=object.Selectable();
   if(selectable_flag)
     {
     //--- set selectable flag of chart object
     object.Selectable(false);
     }
  }
```

Description (Get Method)

Gets a description (text) graphic object.

```
string  Description() const
```

Return Value

Description (text) graphic object that is bound to an instance of the class. If no bound object, it returns NULL.

Description (Set Method)

Sets the description (text) graphic object.

```
bool  Description(
   string  text      // Text
   )
```

Parameters

text

[in] New value descriptions (text) graphic object.

Return Value

true - if successful, false - if you can not change the description (text).

Example:

```
//--- example for CChartObject::Description
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get description of chart object
   string description=object.Description();
   if(description=="")
     {
      //--- set description of chart object
      object.Description("MyObject");
     }
  }
```

Tooltip (Get Method)

Gets the text of the tooltip of graphic object.

```
string  Tooltip() const
```

Returned value

The text of a tooltip. If the object is not assigned, it returns NULL.

Tooltip (Set Method)

Sets the text of a tooltip.

```
bool  Tooltip(
   string  new_tooltip      // new text of a tooltip
   )
```

Parameters

new_tooltip

[in] New value of a tooltip

Returned value

true - if successful, false if tooltip change has failed.

Note:

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.

Timeframes (Get Method)

Gets the visibility flag graphic object.

```
int  Timeframes() const
```

Return Value

Flags visibility graphic object that is bound to an instance of the class. If not bound object, it returns 0.

Timeframes (Set Method)

Sets the visibility flag graphic.

```
bool  Timeframes(
   int  new_timeframes      // Visibility flags
   )
```

Parameters

new_timeframes

[in] New flags visibility graphic object.

Return Value

true - if successful, false - if you can not change the flags of visibility.

Example:

```
//--- example for CChartObject::Timeframes
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get timeframes of chart object
   int timeframes=object.Timeframes();
   if(!(timeframes&OBJ_PERIOD_H1))
     {
      //--- set timeframes of chart object
      object.Timeframes(timeframes|OBJ_PERIOD_H1);
     }
  }
```

Z_Order (Get Method)

Gets the value of priority of graphical object of clicking on a chart (CHARTEVENT_CLICK).

```
long  Z_Order() const
```

Return Value

Priority of a graphic object, assigned to the class instance. If there is no object assigned, it returns 0.

Z_Order (Set Method)

Sets priority of graphical object of clicking on a chart (CHARTEVENT_CLICK).

```
bool  Z_Order(
   long  value      // new priority
   )
```

Parameters

value

[in] New value of priority of a graphical object of clicking on a chart (CHARTEVENT_CLICK).

Return Value

true - if successful, false - if you can not change the priority.

Note

Z_Order is a priority of a graphical object for receiving events of clicking on a chart (CHARTEVENT_CLICK). By setting the value, greater than 0(default value) you can increase the object priority.

CreateTime

Gets the time to create graphical object.

```
datetime  CreateTime() const
```

Return Value

Time to create graphical object that is bound to an instance of the class. If not bound object, it returns 0.

Example:

```
//--- example for CChartObject::CreateTime
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get create time of chart object
   datetime create_time=object.CreateTime();
  }
```

LevelsCount (Get Method)

Gets the number of levels of graphical object.

```
int  LevelsCount() const
```

Return Value

Number of levels of graphical object that is bound to an instance of the class. If not bound object, it returns 0.

LevelsCount (Set Method)

Sets the number of levels of graphical object.

```
bool  LevelsCount(
   int  levels      // Number of levels
   )
```

Parameters

levels

[in] the number of new levels of graphic object.

Return Value

true - if successful, false - if you can not change the number of levels.

Example:

```
//--- example for CChartObject::LevelsCount
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get levels count of chart object
   int levels_count=object.LevelsCount();
   //--- set levels count of chart object
   object.LevelsCount(levels_count+1);
  }
```

LevelColor (Get Method)

Gets the line color specified level of graphic object.

```
color  LevelColor(
   int  level      // Level number
   ) const
```

Parameters

level

[in] Number of levels of graphical object.

Return Value

Color Line level specified graphic object that is bound to an instance of the class. If not bound object or the object is no specified level, it returns CLR_NONE.

LevelColor (Set Method)

Sets the line color specified level of graphic object.

```
bool  LevelColor(
   int    level,          // Level number
   color  new_color       // New color
   )
```

Parameters

level

[in] Number of levels of graphical object.

new_color

[in] New value line color of the level of graphic object.

Return Value

true - if successful, false - if you can not change the color.

Example:

```
//--- example for CChartObject::LevelColor
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level color of chart object
      color level_color=object.LevelColor(i);
      if(level_color!=clrRed)
```

```
         {
          //--- set level color of chart object
          object.LevelColor(i,clrRed);
         }
      }
   }
```

LevelStyle (Get Method)

Gets the line style specified level of graphical object.

```
ENUM_LINE_STYLE  LevelStyle(
   int  level      // Level number
   ) const
```

Parameters

level

[in] Number of levels of graphical object.

Return Value

Line style specified level graphical object that is bound to an instance of the class. If not bound object or the object is no specified level of returns WRONG_VALUE.

LevelStyle (Set Method)

Sets the line style specified level of graphical object.

```
int  LevelStyle(
   int              level,     // Level number
   ENUM_LINE_STYLE  style      // Line Style
   )
```

Parameters

level

[in] Number of levels of graphical object.

style

[in] New value-style line level specified graphical object.

Return Value

true - if successful, false - if you can not change the style.

Example:

```
//--- example for CChartObject::LevelStyle
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level style of chart object
      ENUM_LINE_STYLE level_style=object.LevelStyle(i);
      if(level_style!=STYLE_SOLID)
```

```
         {
          //--- set level style of chart object
          object.LevelStyle(i,STYLE_SOLID);
         }
      }
   }
```

LevelWidth (Get Method)

Gets the line thickness specified level of graphic object.

```
int  LevelWidth(
   int  level      // Level number
   ) const
```

Parameters

level

[in] Number of levels of graphical object.

Return Value

The thickness of the lines of the level of graphical object that is bound to an instance of the class. If not bound object or the object is no specified level, it returns -1.

LevelWidth (Set Method)

Finds the last element equal to the model in sorted array.

```
bool  LevelWidth(
   int  level,          // Level number
   int  new_width       // New width
   )
```

Parameters

level

[in] Number of levels of graphical object.

new_width

[in] New value line thickness specified level graphical object.

Return Value

position of the found element, if successful, -1 - if the item was not found.

Example:

```
//--- example for CChartObject::LevelWidth
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level width of chart object
      int level_width=object.LevelWidth(i);
      if(level_width!=1)
```

```
         {
          //--- set level width of chart object
          object.LevelWidth(i,1);
         }
      }
   }
```

LevelValue (Get Method)

Gets the value of the level of graphic object.

```
double  LevelValue(
   int  level      // Level number
   ) const
```

Parameters

level

[in] Number of levels of graphical object.

Return Value

The value of this level of graphical object that is bound to an instance of the class. If not bound object or the object is no level specified, returns EMPTY_VALUE.

LevelValue (Set Method)

Sets the value of the specified level of graphic object.

```
bool  LevelValue(
   int     level,          // Level number
   double  new_value       // New value
   )
```

Parameters

level

[in] Number of levels of graphical object.

new_value

[in] New value of the level of graphic object.

Return Value

true - if successful, false - if you can not change the level.

Example:

```
//--- example for CChartObject::LevelValue
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level value of chart object
      double level_value=object.LevelValue(i);
      if(level_value!=0.1*i)
```

```
         {
          //--- set level value of chart object
          object.LevelValue(i,0.1*i);
         }
      }
   }
```

LevelDescription (Get Method)

Gets a description (text) of the level of graphical object.

```
string  LevelDescription(
   int  level       // Level number
   ) const
```

Parameters

level

[in] Number of levels of graphical object.

Return Value

Description (text) of the level of graphical object that is bound to an instance of the class. If not bound object or the object is no specified level, returns NULL.

LevelDescription (Set Method)

Sets the description (text) of the level of graphical object.

```
bool  LevelDescription(
   int     level ,     // Level number
   string  text        // Text
   )
```

Parameters

level

[in] Number of level of graphical object.

text

[in] New value of description (text) of the level of graphic object.

Return Value

true - if successful, false - if you can not change the description (text).

Example:

```
//--- example for CChartObject::LevelDescription
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //---
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get level description of chart object
      string level_description=object.LevelDescription(i);
      if(level_description=="")
```

```
         {
          //--- set level description of chart object
          object.LevelDescription(i,"Level_"+IntegerToString(i));
         }
      }
   }
```

GetInteger

Provides simplified access to the functions of API MQL5 ObjectGetInteger() for integer-value properties (of type bool, char, uchar, short, ushort, int, uint, ling, ulong, datetime, color) bound to an instance of the class graphic.There are two versions of a function call:

Getting a property value without checking the correctness

```
long  GetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER  prop_id,          // Identifier of integer-property
   int                           modifier=-1       // Modifier
   ) const
```

Parameters

prop_id

[in] ID of double-graphic properties.

modifier=-1

[in] Modifier (index) double-features.

Return Value

If successfull, it returns the value of integer-type property, if error, it returns 0.

Getting a property value in verifying the correctness of such treatment

```
bool  GetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER  prop_id,       // Identifier of integer-property
   int                           modifier,      // Modifier
   long&                         value          // Link to variable
   ) const
```

Parameters

prop_id

[in] ID integer-graphic properties of the object.

modifier

[in] Modifier (index) integer-property.

value

[out] Reference to a variable to accommodate the integer-value properties.

Return Value

true - if successful, false - if you can not get integer-property.

Example:

```
//--- example for CChartObject::GetInteger
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
```

```
{
 CChartObject object;
 //--- get color of chart object by easy method
 printf("Objects color is %s",ColorToString(object.GetInteger(OBJPROP_COLOR),true))
 //--- get color of chart object by classic method
 long color_value;
 if(!object.GetInteger(OBJPROP_COLOR,0,color_value))
   {
    printf("Get integer property error %d",GetLastError());
    return;
   }
 else
    printf("Objects color is %s",color_value);
 for(int i=0;i<object.LevelsCount();i++)
   {
    //--- get levels width by easy method
    printf("Level %d width is %d",i,object.GetInteger(OBJPROP_LEVELWIDTH,i));
    //--- get levels width by classic method
    long width_value;
    if(!object.GetInteger(OBJPROP_LEVELWIDTH,i,width_value))
      {
       printf("Get integer property error %d",GetLastError());
       return;
      }
    else
       printf("Level %d width is %d",i,width_value);
   }
}
```

SetInteger

Provides simplified access to the functions of API MQL5 ObjectSetInteger() to change the integer-properties (with types bool, char, uchar, short, ushort, int, uint, ling, ulong, datetime, color) bound to an instance of the class graphic.There are two versions of a function call:

Setting a property value that does not require a modifier

```
bool  SetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER   prop_id,      // Identifier of integer-property
   long                           value         // Value
   )
```

Parameters

prop_id

[in] ID integer-graphic properties of the object.

value

[in] new mutable integer-value properties.

Setting a property value indicating the modifier

```
bool  SetInteger(
   ENUM_OBJECT_PROPERTY_INTEGER   prop_id,      // Identifier of integer-property
   int                            modifier,     // Modifier
   long                           value         // Value
   )
```

Parameters

prop_id

[in] ID integer-graphic properties of the object.

modifier

[in] Modifier (index) integer-property.

value

[in] new mutable integer-value properties.

Return Value

true - if successful, false - if you can not change the integer-property.

Example:

```
//--- example for CChartObject::SetInteger
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- set new color of chart object
```

```
   if(!object.SetInteger(OBJPROP_COLOR,clrRed))
     {
      printf("Set integer property error %d",GetLastError());
      return;
     }
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- set levels width
      if(!object.SetInteger(OBJPROP_LEVELWIDTH,i,i))
        {
         printf("Set integer property error %d",GetLastError());
         return;
        }
     }
  }
```

GetDouble

Provides simplified access to the functions of API MQL5 ObjectGetDouble() to get the values double-properties (having type float and double) of the graphic object, assigned to the class instance. There are two versions of a function call:

Getting a property value without checking the correctness

```
double  GetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE  prop_id,          // Identifier of double-property
   int                          modifier=-1      // Modifier
   ) const
```

Parameters

prop_id

[in] ID of double-graphic properties.

modifier=-1

[in] Modifier (index) double-features.

Return Value

If successful, it returns the value of property of double type, if error, it returns EMPTY_VALUE.

Getting a property value in verifying the correctness of such treatment

```
bool  GetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE  prop_id,       // Identifier of double-property
   int                          modifier,      // Modifier
   double&                      value          // Link to variable
   ) const
```

Parameters

prop_id

[in] ID of double-graphic properties.

modifier

[in] Modifier (index) double-features.

value

[out] Reference to a variable to accommodate the double-value properties.

Return Value

true - if successful, false - if you can not get a double-feature.

Example:

```
//--- example for CChartObject::GetDouble
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
```

```
{
 CChartObject object;
 //---
 for(int i=0;i<object.LevelsCount();i++)
   {
    //--- get levels value by easy method
    printf("Level %d value=%f",i,object.GetDouble(OBJPROP_LEVELVALUE,i));
    //--- get levels value by classic method
    double value;
    if(!object.SetDouble(OBJPROP_LEVELVALUE,i,value))
      {
       printf("Get double property error %d",GetLastError());
       return;
      }
    else
       printf("Level %d value=%f",i,value);
   }
}
```

SetDouble

Provides simplified access to the functions of API MQL5 ObjectSetDouble() to change the double-properties (having type float and double) bound to an instance of the class graphic object. There are two versions of a function call:

Setting a property value that does not require a modifier

```
bool  SetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,      // Identifier of double-property
   double                        value         // Value
   )
```

Parameters

prop_id

[in] ID of double-graphic properties.

value

[in] New value mutable double-features.

Setting a property value indicating the modifier

```
bool  SetDouble(
   ENUM_OBJECT_PROPERTY_DOUBLE   prop_id,       // Identifier of double-property
   int                           modifier,      // Modifier
   double                        value          // Value
   )
```

Parameters

prop_id

[in] ID of double-graphic properties.

modifier

[in] Modifier (index) of double-property.

value

[in] New value mutable double-property.

Return Value

true - if successful, false - if you can not change the double-feature.

Example:

```
//--- example for CChartObject::SetDouble
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
//---
```

```
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- set level value of chart object
      if(!object.SetDouble(OBJPROP_LEVELVALUE,i,0.1*i))
        {
         printf("Set double property error %d",GetLastError());
         return;
        }
     }
  }
```

GetString

Provides simplified access to the functions of API MQL5 ObjectGetString() for string-value properties, is bound to an instance of the class graphic object. There are two versions of a function call:

Getting a property value without checking the correctness

```
string  GetString(
   ENUM_OBJECT_PROPERTY_STRING  prop_id,          // Identifier of string-property
   int                          modifier=-1      // Modifier
   ) const
```

Parameters

prop_id

[in] Identifier string-properties of graphic object.

modifier=-1

[in] Modifier (index) string-properties.

Return Value

Value of string-property.

Getting a property value in verifying the correctness of such treatment

```
bool  GetString(
   ENUM_OBJECT_PROPERTY_STRING  prop_id,       // Identifier of string-property
   int                          modifier,      // Modifier
   string&                      value          // Link to variable
   ) const
```

Parameters

prop_id

[in] Identifier string-properties of graphic object.

modifier

[in] Modifier (index) string-properties.

value

[out] Reference to a variable to accommodate the string-value properties.

Return Value

true - if successful, false - if you can not get a string-property.

Example:

```
//--- example for CChartObject::GetString
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
```

```
   CChartObject object;
   string       value;
   //--- get name of chart object by easy method
   printf("Object name is '%s'",object.GetString(OBJPROP_NAME));
   //--- get name of chart object by classic method
   if(!object.GetString(OBJPROP_NAME,0,value))
     {
      printf("Get string property error %d",GetLastError());
      return;
     }
   else
      printf("Object name is '%s'",value);
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- get levels description by easy method
      printf("Level %d description is '%s'",i,object.GetString(OBJPROP_LEVELTEXT,i));
      //--- get levels description by classic method
      if(!object.GetString(OBJPROP_LEVELTEXT,i,value))
        {
         printf("Get string property error %d",GetLastError());
         return;
        }
      else
         printf("Level %d description is '%s'",i,value);
     }
  }
```

SetString

Provides simplified access to the functions of API MQL5 ObjectSetString() to change the properties of string-tied to the instance of the class graphic object. There are two versions of a function call:

Setting a property value that does not require a modifier

```
bool  SetString(
   ENUM_OBJECT_PROPERTY_STRING   prop_id,     // Identifier of string-property
   string                        value        // Value
   )
```

Parameters

prop_id

[in] Identifier string-properties of graphic object.

value

[in] New value mutable string-properties.

Setting a property value indicating the modifier

```
bool  SetString(
   ENUM_OBJECT_PROPERTY_STRING   prop_id,      // Identifier of string-property
   int                           modifier,     // Modifier
   string                        value         // Value
   )
```

Parameters

prop_id

[in] Identifier string-properties of graphic object.

modifier

[in] Modifier (index) string-properties.

value

[in] New value mutable string-properties.

Return Value

true - if successful, false - if you can not change the string-property.

Example:

```
//--- example for CChartObject::SetString
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- set new name of chart object
   if(!object.SetString(OBJPROP_NAME,"MyObject"))
```

```
     {
      printf("Set string property error %d",GetLastError());
      return;
     }
   for(int i=0;i<object.LevelsCount();i++)
     {
      //--- set levels description
      if(!object.SetString(OBJPROP_LEVELTEXT,i,"Level_"+IntegerToString(i)))
        {
         printf("Set string property error %d",GetLastError());
         return;
        }
     }
  }
```

Save

Saves parameters of the object in the file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the previously opened using the function FileOpen (...) file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CChartObject::Save
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   int          file_handle;
   CChartObject object=new CChartObject;
   //--- set object parameters
   //--- . . .
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
  }
```

Load

Loads the parameters of the object from the file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the previously opened using the function FileOpen (...) file.

Return Value

true - if successfully completed, false - if an error.

Example:

```
//--- example for CChartObject::Load
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   int           file_handle;
   CChartObject object;
   //--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!object.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
   //--- use object
   //--- . . .
  }
```

Type

Gets the type identifier graphic object.

```
virtual int  Type() const
```

Return Value

Object type identifier (0x8888 for CChartObject).

Example:

```
//--- example for CChartObject::Type
#include <ChartObjects\ChartObject.mqh>
//---
void OnStart()
  {
   CChartObject object;
   //--- get object type
   int type=object.Type();
  }
```

Objects Lines

A group of graphic objects "Lines".

This section contains the technical details of working with a group of classes of graphical objects "Lines" and a description of the relevant components of the MQL5 Standard Library .

Class name	Object
CChartObjectVLine	Graphic object "Vertical Line"
CChartObjectHLine	Graphic object "Horizontal Line"
CChartObjectTrend	Graphic object "Trend Line"
CChartObjectTrendByAngle	Graphic object "Trend Line by Angle"
CChartObjectCycles	Graphic object "Cyclic Lines"

See also

Object types, Graphic objects

CChartObjectVLine

Class CChartObjectVLine is a class for simplified access to "Vertical Line" graphic object properties.

Description

Class CChartObjectVLine provides access to "Vertical Line" object properties.

Declaration

```
class CChartObjectVLine : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Vertical Line"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Vertical Line".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time           // Time coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time

[in] Time coordinate of the anchor point.

Returned value

true if successful, false if error.

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_VLINE for CChartObjectVLine).

CChartObjectHLine

Class CChartObjectHLine is a class for simplified access to "Horizontal Line" graphic object properties.

Description

Class CChartObjectHLine provides access to "Horizonal Line" object properties.

Declaration

```
class CChartObjectHLine : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Horizontal Line"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Horizontal Line".

```
bool  Create(
   long    chart_id,      // Chart identifier
   string  name,          // Object name
   long    window,        // Chart window
   double  price          // Price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

price

[in] Price coordinate of the anchor point.

Returned value

true if successful, false if error.

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_HLINE for CChartObjectHLine).

CChartObjectTrend

Class CChartObjectTrend is a class for simplified access to "Trend Line" graphic object properties.

Description

Class CChartObjectTrend provides access to "Trend Line" object properties.

Declaration

```
class CChartObjectTrend : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Trend Line"
Properties	
RayLeft	Gets/Sets property "Ray Left"
RayRight	Gets/Sets property "Ray Right"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Derived classes:

- CChartObjectChannel
- CChartObjectFibo
- CChartObjectFiboChannel
- CChartObjectFiboExpansion
- CChartObjectGannFan
- CChartObjectGannGrid
- CChartObjectPitchfork
- CChartObjectRegression
- CChartObjectStdDevChannel
- CChartObjectTrendByAngle

See also

Object types, Graphic objects

Create

Creates graphic object "Trend Line".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // 1st time coordinate
   double    price1,        // 1st price coordinate
   datetime  time2,         // 2nd time coordinate
   double    price2         // 2nd price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Returned value

true if successful, false if error.

RayLeft (Get Method)

Gets the value of "Ray Left" property.

```
bool  RayLeft() const
```

Returned value

The value of "Ray Left" property, assigned to the class instance. If there is no object assigned, it returns false.

RayLeft (Set Method)

Sets new flag value for the "Ray Left" property.

```
bool  RayLeft(
   bool  ray      // flag
   )
```

Parameters

ray

[in] New value of the "Ray Left" property.

Returned value

true if successful, false if flag hasn't changed error.

RayRight (Get Method)

Gets the value of "Ray Right" property.

```
bool  RayRight() const
```

Returned value

The value of "Ray Right" property, assigned to the class instance. If there is no object assigned, it returns false.

RayRight (Set Method)

Sets new flag value for the "Ray Right" property.

```
bool  RayRight(
   bool  ray      // flag
   )
```

Parameters

ray

[in] New value of the "Ray Right" property.

Returned value

true if successful, false if flag hasn't changed error.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the file already opened using by function FileOpen(...) .

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_TREND for CChartObjectTrend).

CChartObjectTrendByAngle

Class CChartObjectTrendByAngle is a class for simplified access to "Trend Line by Angle" graphic object properties.

Description

Class CChartObjectTrendByAngle provides access to "Trend Line by Angle" object properties.

Declaration

```
class CChartObjectTrendByAngle : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Trend Line by Angle"
Properties	
Angle	Gets/Sets property "Angle"
Input/output	
virtual Type	Virtual method of identification

Derived classes:

- CChartObjectGannLine

See also

Object types, Graphic objects

Create

Creates graphic object "Trend Line by Angle"

```
bool  Create(
   long      chart_id,     // Chart identifier
   string    name,         // Object name
   long      window,       // Chart window
   datetime  time1,        // 1st time coordinate
   double    price1,       // 1st price coordinate
   datetime  time2,        // 2nd time coordinate
   double    price2        // 2nd price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Returned value

true if successful, false if error.

Angle (Get Method)

Gets the value of "Angle" property.

```
double  Angle() const
```

Returned value

The value of "Angle" property, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Angle (Set Method)

Sets new value for the "Angle" property.

```
bool  Angle(
   double  angle      // Angle
   )
```

Parameters

angle

[in] New value of the "Angle" property.

Returned value

true if successful, false if property hasn't changed.

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_TRENDBYANGLE for CChartObjectTrendByAngle).

CChartObjectCycles

Class CChartObjectCycles is a class for simplified access to "Cyclic Lines" graphic object properties.

Description

Class CChartObjectCycles provides access to "Cyclic Lines" object properties.

Declaration

```
class CChartObjectCycles : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsLines.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Cycle Lines"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Cyclic Lines"

```
bool  Create(
   long      chart_id,     // Chart identifier
   string    name,         // Object name
   long      window,       // Chart window
   datetime  time1,        // 1st time coordinate
   double    price1,       // 1st price coordinate
   datetime  time2,        // 2nd time coordinate
   double    price2        // 2nd price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Returned value

true if successful, false if error.

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_CYCLES for CChartObjectCycles).

Objects Channels

A group of graphic objects "Channels".

This section contains the technical details of working with a group of classes of graphical objects "Channels" and a description of the relevant components of the MQL5 Standard Library .

Class name	Object
CChartObjectChannel	Graphic object "Equidistant Channel"
CChartObjectRegression	Graphic object "Linear Regression Channel"
CChartObjectStdDevChannel	Graphic object "Standard deviations Channel"
CChartObjectPitchfork	Graphic object "Andrew's Pitchfork"

See also

Object types, Graphic objects

CChartObjectChannel

The CChartObjectChannel is a class for simplified access to "Equidistant Channel" graphic object properties.

Description

The class CChartObjectChannel provides access to "Equidistant Channel" object properties.

Declaration

```
class CChartObjectChannel : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Equidistant Channel"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Equidistant Channel"

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // Time coordinate
   double    price1,
   datetime  time2,
   double    price2,
   datetime  time3,
   double    price3
   )
```

Parameters

chart_id

[in] Chart ID (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Returned value

true if successful, false if error.

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_CHANNEL for CChartObjectChannel).

CChartObjectRegression

Class CChartObjectRegression is a class for simplified access to "Linear Regression Channel" graphic object properties.

Description

Class CChartObjectRegression provides access to "Linear Regression Channel" object properties.

Declaration

```
class CChartObjectRegression : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Linear Regression Channel"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Linear Regression Channel"

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   long      window,        // Chart window
   datetime  time1,         // First time coordinate
   datetime  time2          // Second time coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

Returned value

true if successful, false if error.

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_REGRESSION for CChartObjectRegression).

CChartObjectStdDevChannel

Class CChartObjectStdDevChannel is a class for simplified access to "Standard Deviation Channel" graphic object properties.

Description

Class CChartObjectStdDevChannel provides access to "Standard Deviation Channel" object properties.

Declaration

```
class CChartObjectStdDevChannel : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Standard Deviation Channel"
Properties	
Deviations	Gets/Sets property "Deviation"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Standard Deviation Channel"

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   datetime  time2,         // Second time coordinate
   double    deviation      // Deviation
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

deviation

[in] Numerical value for "Deviation" property.

Returned value

true if successful, false if error.

Deviation (Get Method)

Gets numerical value for "Deviation" property.

```
double  Deviation() const
```

Returned value

Numerical value of "Deviation" property, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Deviation (Set Method)

Sets numerical value for "Deviation" property.

```
bool  Deviation(
   double  deviation      // Deviation
   )
```

Parameters

deviation

[in] New value for "Deviation" property.

Returned value

true if successful, false if property hasn't changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the file already opened using by function FileOpen(...) .

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the file already opened using by function FileOpen(...) .

Returned value

true if successful, false if error.

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_STDDEVCHANNEL for CChartObjectStdDevChannel).

CChartObjectPitchfork

Class CChartObjectPitchfork is a class for simplified access to "Andrew's Pitchfork" graphic object properties.

Description

Class CChartObjectPitchfork provides access to "Andrew's Pitchfork" object properties.

Declaration

```
class CChartObjectPitchfork : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsChannels.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Andrew's Pitchfork"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Andrew's Pitchfork"

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   long      window,        // Chart window
   datetime  time1,         // 1st time coordinate
   double    price1,        // 1st price coordinate
   datetime  time2,         // ...
   double    price2,
   datetime  time3,
   double    price3
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point..

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the first anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Returned value

true if successful, false if error.

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_PITCHFORK for CChartObjectPitchfork).

Gann Tools

A group of graphic objects "Gann Tools".

This section contains the technical details of working with a group of classes of graphical objects "Gann Tools" and a description of the relevant components of the MQL5 Standard Library .

Class name	Object
CChartObjectGannLine	Graphic object "Gann Line"
CChartObjectGannFan	Graphic object "Gann Fan"
CChartObjectGannGrid	Graphic object "Gann Grid"

See also

Object types, Graphic objects

CChartObjectGannLine

Class CChartObjectGannLine is a class for simplified access to "Gann Line" graphic object properties.

Description

Class CChartObjectGannLine provides access to "Gann Line" object properties.

Declaration

```
class CChartObjectGannLine : public CChartObjectTrendByAngle
```

Title

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Gann Line"
Properties	
PipsPerBar	Gets/Sets property "Scale"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Gann Line".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    ppb            // Pips per bar
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

ppb

[in] Pips per bar.

Returned value

true if successful, false if error.

PipsPerBar (Get Method)

Gets the value of "Pips per bar" property.

```
double  PipsPerBar() const
```

Returned value

Value of "Pips per bar" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

PipsPerBar (Set Method)

Sets new value for "Pips per bar" property.

```
bool  PipsPerBar(
   double  ppb      // Pips per bar
   )
```

Parameters

ppb

[in] New value for "Pips per bar" property.

Returned value

true if successful, false if property hasn't changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_GANNLINE for CChartObjectGannLine).

CChartObjectGannFan

Class CChartObjectGannFan is a class for simplified access to "Gann Fan" graphic object properties.

Description

Class CChartObjectGannFan provides access to "Gann Fan" object properties.

Declaration

```
class CChartObjectGannFan : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Gann Fan"
Properties	
PipsPerBar	Gets/Sets property "Pips per bar"
Downtrend	Gets/Sets property "Downtrend"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Gann Fan".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    ppb            // Pips per bar
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

ppb

[in] Pips per bar.

Returned value

true if successful, false if error.

PipsPerBar (Get Method)

Gets the value of "Pips per bar" property.

```
double  PipsPerBar() const
```

Returned value

Value of "Pips per bar" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

PipsPerBar (Set Method)

Sets new value for "Pips per bar" property.

```
bool  PipsPerBar(
   double  ppb      // Pips per bar
   )
```

Parameters

ppb

[in] New value for "Pips per bar" property.

Returned value

true if successful, false if property hasn't changed.

Downtrend (Get Method)

Gets the value of "Downtrend" property.

```
bool  Downtrend() const
```

Returned value

Value of the "Downtrend" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

Downtrend (Set Method)

Sets new value of "Downtrend" property.

```
bool  Downtrend(
   bool  downtrend      // Flag value
   )
```

Parameters

downtrend

[in] New value for "Downtrend" property.

Returned value

true if successful, false if property hasn't changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_GANNFAN for CChartObjectGannFan).

CChartObjectGannGrid

Class CChartObjectGannGrid is a class for simplified access to "Gann Grid" graphic object properties.

Description

Class CChartObjectGannGrid provides access to "Gann Grid" object properties.

Declaration

```
class CChartObjectGannGrid : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsGann.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Gann Grid"
Properties	
PipsPerBar	Gets/Sets property "Pips per bar"
Downtrend	Gets/Sets property "Downtrend"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Gann Grid".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    ppb            // Pips per bar
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

ppb

[in] Pips per bar.

Returned value

true if successful, false if error.

PipsPerBar (Get Method)

Gets the value of "Pips per bar" property.

```
double  PipsPerBar() const
```

Returned value

Value of "Pips per bar" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

PipsPerBar (Set Method)

Sets new value for "Pips per bar" property.

```
bool  PipsPerBar(
   double  ppb      // Pips per bar
   )
```

Parameters

ppb

[in] New value for "Pips per bar" property.

Returned value

true if successful, false if property hasn't changed.

Downtrend (Get Method)

Gets the value of "Downtrend" property.

```
bool  Downtrend() const
```

Returned value

Value of "Downtrend" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

Downtrend (Set Method)

Sets new value of "Downtrend" property.

```
bool  Downtrend(
   bool  downtrend      // Flag value
   )
```

Parameters

downtrend

[in] New value for "Downtrend" property.

Returned value

true if successful, false if property hasn't changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_GANNGRID for CChartObjectGannGrid).

Fibonacci Tools

A group of graphic objects "Fibonacci Tools".

This section contains the technical details of working with a group of classes of graphical objects "Fibonacci Tools" and a description of the relevant components of the MQL5 Standard Library .

Class name	Object
CChartObjectFibo	Graphic object "Fibonacci Retracement"
CChartObjectFiboTimes	Graphic object "Fibonacci Time Zones"
CChartObjectFiboFan	Graphic object "Fibonacci Fan"
CChartObjectFiboArc	Graphic object "Fibonacci Arc"
CChartObjectFiboChannel	Graphic object "Fibonacci Channel"
CChartObjectFiboExpansion	Graphic object "Fibonacci Expansion"

See also

Object types, Graphic objects

CChartObjectFibo

Class CChartObjectFibo is a class for simplified access to "Fibonacci Retracement" graphic object properties.

Description

Class CChartObjectFibo provides access to "Fibonacci Retracement" object properties.

Declaration

```
class CChartObjectFibo : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Fibonacci Retracement"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Fibonacci Retracement".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    price2         // Second price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_FIBO for CChartObjectFibo).

CChartObjectFiboTimes

Class CChartObjectFiboTimes is a class for simplified access to "Fibonacci Time Zones" graphic object properties.

Description

Class CChartObjectFiboTimes provides access to "Fibonacci Time Zones" object properties.

Declaration

```
class CChartObjectFiboTimes : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Fibonacci Time Zones"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Fibonacci Time Zones".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    price2         // Second price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_FIBOTIMES for CChartObjectFiboTimes).

CChartObjectFiboFan

Class CChartObjectFiboFan is a class for simplified access to "Fibonacci Fan" graphic object properties.

Description

Class CChartObjectFiboFan provides access to "Fibonacci Fan" object properties.

Declaration

```
class CChartObjectFiboFan : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Fibonacci Fan"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Fibonacci Fan".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    price2         // Second price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_FIBOFAN for CChartObjectFiboFan).

CChartObjectFiboArc

Class CChartObjectFiboArc is a class for simplified access to "Fibonacci Arc" graphic object properties.

Description

Class CChartObjectFiboArc provides access to "Fibonacci Arc" object properties.

Declaration

```
class CChartObjectFiboArc : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Fibonacci Arc"
Properties	
Scale	Gets/Sets property "Scale"
Ellipse	Gets/Sets property "Ellipse"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Fibonacci Arc"

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    price2,        // Second price coordinate
   double    scale          // Scale
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

scale

[in] Scale.

Returned value

true if successful, false if error.

Scale (Get Method)

Gets the value of "Scale" property.

```
double  Scale() const
```

Returned value

Value of "Scale" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Scale (Set Method)

Sets new value for "Scale" property.

```
bool  Scale(
   double  scale      // Scale
   )
```

Parameters

scale

[in] New value for "Scale" property.

Returned value

true if successful, false if property hasn't changed.

Ellipse (Get Method)

Gets the value of "Ellipse" property.

```
bool  Ellipse() const
```

Returned value

Value of "Ellipse" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

Ellipse (Set Method)

Sets new flag value for "Ellipse" property.

```
bool  Ellipse(
   bool  ellipse      // flag value
   )
```

Parameters

ellipse

[in] New value for "Scale" property.

Returned value

true if successful, false if property hasn't changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_FIBOARC for CChartObjectFiboArc).

CChartObjectFiboChannel

Class CChartObjectFiboChannel is a class for simplified access to "Fibonacci Channel" graphic object properties.

Description

Class CChartObjectFiboChannel provides access to "Fibonacci Channel" object properties.

Declaration

```
class CChartObjectFiboChannel : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Fibonacci Channel"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Fibonacci Channel".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    price2,        // Second price coordinate
   datetime  time3,         // Third time coordinate
   double    price3         // Third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_FIBOCHANNEL for CChartObjectFiboChannel).

CChartObjectFiboExpansion

Class CChartObjectFiboExpansion is a class for simplified access to "Fibonacci Expansion" graphic object properties.

Description

Class CChartObjectFiboExpansion provides access to "Fibonacci Expansion" object properties.

Declaration

```
class CChartObjectFiboExpansion : public CChartObjectTrend
```

Title

```
#include <ChartObjects\ChartObjectsFibo.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Fibonacci Expansion"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Fibonacci Expansion".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    price2,        // Second price coordinate
   datetime  time3,         // Third time coordinate
   double    price3         // Third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_EXPANSION for CChartObjectFiboExpansion).

Elliott Tools

A group of graphic objects "Elliott Tools".

This section contains the technical details of working with a group of classes of graphical objects "Elliott Tools" and a description of the relevant components of the MQL5 Standard Library .

Class name	Object
CChartObjectElliottWave3	Graphic object "Correcting Wave"
CChartObjectElliottWave5	Graphic object "Impulse Wave"

See also

Object types, Graphic objects

CChartObjectElliottWave3

Class CChartObjectElliottWave3 is a class for simplified access to "Correcting Wave" graphic object properties.

Description

Class CChartObjectElliottWave3 provides access to "Correcting Wave" object properties.

Declaration

```
class CChartObjectElliottWave3 : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsElliott.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Correcting Wave"
Properties	
Degree	Gets/Sets property "Degree"
Lines	Gets/Sets property "Lines"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Derived classes:

- CChartObjectElliottWave5

See also

Object types, Graphic objects

Create

Creates graphic object "Correcting Wave".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    price2,        // Second price coordinate
   datetime  time3,         // Third time coordinate
   double    price3         // Third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Time coordinate of the third anchor point.

Returned value

true if successful, false if error.

Degree (Get Method)

Gets the value of "Degree" property.

```
ENUM_ELLIOT_WAVE_DEGREE  Degree() const
```

Returned value

Value of "Degree" property of the object, assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Degree (Set Method)

Sets new value for "Degree" property.

```
bool  Degree(
   ENUM_ELLIOT_WAVE_DEGREE  degree      // property value
   )
```

Parameters

degree

[in] New value for "Degree" property.

Returned value

true if successful, false if property hasn't changed.

Lines (Get Method)

Gets the value of "Lines" property.

```
bool  Lines() const
```

Returned value

Value of "Lines" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

Lines (Set Method)

Sets new value for "Lines" property.

```
bool  Lines(
   bool  lines      // flag value
   )
```

Parameters

lines

[in] New value for "Lines" property.

Returned value

true if successful, false if flag hasn't changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened using by function FileOpen(...).

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_ELLIOTWAVE3 for CChartObjectElliottWave3).

CChartObjectElliottWave5

Class CChartObjectElliottWave5 is a class for simplified access to "Impulse Wave" graphic object properties.

Description

Class CChartObjectElliottWave5 provides access to "Impulse Wave" object properties.

Declaration

```
class CChartObjectElliottWave5 : public CChartObjectElliottWave3
```

Title

```
#include <ChartObjects\ChartObjectsElliott.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Impulse Wave"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Impulse Wave".

```
bool  Create(
   long      chart_id,     // Chart identifier
   string    name,         // Object name
   int       window,       // Chart window
   datetime  time1,        // First time coordinate
   double    price1,       // First price coordinate
   datetime  time2,        // Second time coordinate
   double    price2,       // Second price coordinate
   datetime  time3,        // Third time coordinate
   double    price3,       // Third price coordinate
   datetime  time4,        // Fourth time coordinate
   double    price4,       // Fourth price coordinate
   datetime  time5         // Fifth time coordinate
   double    price5,       // Fifth price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

time4

[in] Time coordinate of the fourth anchor point.

price4

[in] Price coordinate of the fourth anchor point.

time5

[in] Time coordinate of the fifth anchor point.

price5

[in] Price coordinate of the fifth anchor point.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_ELLIOTWAVE5 for CChartObjectElliottWave5).

Objects Shapes

A group of graphic objects "Shapes".

This section contains the technical details of working with a group of classes of graphical objects "Shapes" and a description of the relevant components of the MQL5 Standard Library .

Class name	Object
CChartObjectRectangle	Graphic object "Rectangle"
CChartObjectTriangle	Graphic object "Triangle"
CChartObjectEllipse	Graphic object "Ellipse"

See also

Object types, Graphic objects

CChartObjectRectangle

Class CChartObjectRectangle is a class for simplified access to "Rectangle" graphic object properties.

Description

Class CChartObjectRectangle provides access to "Rectangle" object properties.

Declaration

```
class CChartObjectRectangle : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Rectangle"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Rectangle".

```
bool  Create(
   long      chart_id,     // Chart identifier
   string    name,         // Object name
   long      window,       // Chart window
   datetime  time1,        // First time coordinate
   double    price1,       // First price coordinate
   datetime  time2,        // Second time coordinate
   double    price2        // Second price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

Returned value

true if successful, 0 if error

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_RECTANGLE for CChartObjectRectangle).

CChartObjectTriangle

Class CChartObjectTriangle is a class for simplified access to "Triangle" graphic object properties.

Description

Class CChartObjectTriangle provides access to "Triangle" object properties.

Declaration

```
class CChartObjectTriangle : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Triangle"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Triangle".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   long      window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    price2,        // Second price coordinate
   datetime  time3,         // Third time coordinate
   double    price3         // Third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Returned value

true if successful, false if error

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_TRIANGLE for CChartObjectTriangle).

CChartObjectEllipse

Class CChartObjectEllipse is a class for simplified access to "Ellipse" graphic object properties.

Description

Class CChartObjectEllipse provides access to "Ellipse" object properties.

Declaration

```
class CChartObjectEllipse : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsShapes.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Ellipse"
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Graphic objects

Create

Creates graphic object "Ellipse".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time1,         // First time coordinate
   double    price1,        // First price coordinate
   datetime  time2,         // Second time coordinate
   double    price2,        // Second price coordinate
   datetime  time3,         // Third time coordinate
   double    price3         // Third price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time1

[in] Time coordinate of the first anchor point.

price1

[in] Price coordinate of the first anchor point.

time2

[in] Time coordinate of the second anchor point.

price2

[in] Price coordinate of the second anchor point.

time3

[in] Time coordinate of the third anchor point.

price3

[in] Price coordinate of the third anchor point.

Returned value

true if successful, false if error

Type

Returns object type identifier of the graphic object.

```
int  Type() const
```

Returned value

Object type identifier (OBJ_ELLIPSE for CChartObjectEllipse).

Objects Arrows

Group for graphic objects Arrows.

This section contains the technical details of working with a group of classes of graphical objects "Arrow" and a description of the relevant components of the MQL5 Standard Library . In essence, the arrow - this is some icon to be displayed to the user, which matches a certain code. There are two types of graphical objects "Arrow" to display icons on the charts:

- Object "Arrow", which allows you to specify the code icon displayed object.
- Group objects to display certain types of icons (and the corresponding certain fixed code).

Class for the arrow displays icons of arbitrary code

Class name	Name of the object arrow
CChartObjectArrow	Arrow

Classes for the arrow icon fixed code

Class name	Name of the object arrow
CChartObjectArrowCheck	Check
CChartObjectArrowDown	Arrow Up
CChartObjectArrowUp	Arrow Down
CChartObjectArrowStop	Stop Sign
CChartObjectArrowThumbDown	Thumbs Up
CChartObjectArrowThumbUp	Thumbs Down
CChartObjectArrowLeftPrice	Left Price Label
CChartObjectArrowRightPrice	Right Price Label

See also

Object types, Methods of binding sites, Graphic objects

CChartObjectArrow

Class CChartObjectArrow is a class for simplified access to "Arrow" graphic object properties.

Description

Class CChartObjectArrow provides access to common properties of objects "Arrow" to all of its descendants.

Declaration

```
class CChartObjectArrow : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsArrows.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Arrow"
Properties	
ArrowCode	Gets/Sets property "Arrow Code"
Anchor	Gets/Sets property "Anchor"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Methods of binding sites, Graphic objects

Create

Creates graphic object "Arrow".

```
bool  Create(
   long      chart_id,      // Chart ID
   string    name,          // Object Name
   int       window,        // Chart Window
   datetime  time,          // Time
   double    price,         // Price
   char      code           // Arrow Code
   )
```

Parameters

chart_id

[in] Chart Identifier (0 - current chart).

name

[in] Object name (Should be unique).

window

[in] Chart window number (0 - base window).

time

[in] Time coordinate.

price

[in] Price coordinate.

code

[in] "Arrow" code (Wingdings).

Returned value

true - if successful, otherwise false.

Example:

```
//--- example for CChartObjectArrow::Create
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
```

```
      return;
     }
//--- use arrow
//--- . . .
  }
```

ArrowCode (Get Method)

Gets code of the symbol for "Arrow".

```
char  ArrowCode() const
```

Returned value

Symbol code of "Arrow" object, assigned to the class instance. If there is no object assigned, it returns 0.

ArrowCode (Set Method)

Sets symbol code for "Arrow"

```
bool  ArrowCode(
   char  code      // Code value
   )
```

Parameters

code

[in] new value for "arrow" code (Wingdings).

Returned value

true - if successful, false - if code hasn't changed.

Example:

```
//--- example for CChartObjectArrow::ArrowCode
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
   char              code=181;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,code))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
      return;
     }
//--- use arrow to possible changes code
//--- . . .
//--- get code of arrow
   if(arrow.ArrowCode()!=code)
     {
     //--- set code of arrow
```

```
      arrow.ArrowCode(code);
      }
//--- use arrow
//--- . . .
   }
```

Anchor (Get Method)

Gets anchor type of the "Arrow" object

```
ENUM_ARROW_ANCHOR  Anchor() const
```

Returned value

Anchor type of "Arrow" object, assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Anchor (Set Method)

Sets color for graphic object

```
bool  Anchor(
   ENUM_ARROW_ANCHOR  new_color      // new color
   )
```

Parameters

new_color

[in] New value of color for line of the graphic object.

Returned value

true if successful, false if color hasn't changed.

Example:

```
//--- example for CChartObject::Anchor
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
   ENUM_ARROW_ANCHOR anchor=ANCHOR_BOTTOM;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
      return;
     }
//--- get anchor of arrow
   if(arrow.Anchor()!=anchor)
     {
     //--- set anchor of arrow
     arrow.Anchor(anchor);
     }
```

```
//--- use arrow
//--- . . .
  }
```

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the file already opened using by function FileOpen(...) .

Returned value

true if successful, otherwise false.

Example:

```
//--- example for CChartObjectArrow::Save
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   int               file_handle;
   CChartObjectArrow arrow;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"Arrow",0,TimeCurrent(),price,181))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
      return;
     }
//--- open file
   file_handle=FileOpen("MyFile.bin",FILE_WRITE|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!arrow.Save(file_handle))
        {
         //--- file save error
         printf("File save: Error %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
  }
```

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // file handle
   )
```

Parameters

file_handle

[in] handle of the file already opened using by function FileOpen(...).

Returned value

true if successful, otherwise false.

Example:

```
//--- example for CChartObjectArrow::Load
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   int               file_handle;
   CChartObjectArrow arrow;
//--- open file
   file_handle=FileOpen("MyFile.bin",FILE_READ|FILE_BIN|FILE_ANSI);
   if(file_handle>=0)
     {
      if(!arrow.Load(file_handle))
        {
         //--- file load error
         printf("File load: Error %d!",GetLastError());
         FileClose(file_handle);
         //---
         return;
        }
      FileClose(file_handle);
     }
//--- use arrow
//--- . . .
  }
```

Type

Returns graphic object type identifier

```
virtual int  Type() const
```

Returned value

Object type identifier (for example, OBJ_ARROW for CChartObjectArrow)

Example:

```
//--- example for CChartObjectArrow::Type
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
   CChartObjectArrow arrow;
   //--- get arrow type
   int type=arrow.Type();
  }
```

Arrows with fixed code

Classes "Arrows with fixed code" are classes for simplified access to the properties of the following graphic objects:

Class name	Arrow object name
CChartObjectArrowCheck	"Arrow Check"
CChartObjectArrowDown	"Arrow Down"
CChartObjectArrowUp	"Arrow Up"
CChartObjectArrowStop	"Arrow Stop"
CChartObjectArrowThumbDown	"Good" ("Big finger up")
CChartObjectArrowThumbUp	"Bad" ("Big finger down")
CChartObjectArrowLeftPrice	"Left price" arrow
CChartObjectArrowRightPrice	"Right price" arrow

Description

Classes "Arrows with fixed code" provides access to the object properties.

Declarations

```
class CChartObjectArrowCheck      : public CChartObjectArrow;
class CChartObjectArrowDown       : public CChartObjectArrow;
class CChartObjectArrowUp         : public CChartObjectArrow;
class CChartObjectArrowStop       : public CChartObjectArrow;
class CChartObjectArrowThumbDown  : public CChartObjectArrow;
class CChartObjectArrowThumbUp    : public CChartObjectArrow;
class CChartObjectArrowLeftPrice  : public CChartObjectArrow;
class CChartObjectArrowRightPrice : public CChartObjectArrow;
```

Title

```
<ChartObjects\ChartObjectsArrows.mqh>
```

Class Methods

Create	
Create	Creates graphic object specified
Properties	
ArrowCode	"Gag" for method of code change
Input/output	
virtual Type	Virtual method of identification

See also

Object types, Methods of binding sites, Graphic objects

Create

Creates graphic object "Arrow with fixed code".

```
bool  Create(
   long      chart_id,      // Chart ID
   string    name,          // Object Name
   int       window,        // Chart Window
   datetime  time,          // Time
   double    price          // Price
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] Unique name of the object to create.

window

[in] Chart window number (0 - base window).

time

[in] Time coordinate.

price

[in] Price coordinate.

Returned value

true if successful, false if error.

Example:

```
//--- example for CChartObjectArrowCheck::Create
//--- example for CChartObjectArrowDown::Create
//--- example for CChartObjectArrowUp::Create
//--- example for CChartObjectArrowStop::Create
//--- example for CChartObjectArrowThumbDown::Create
//--- example for CChartObjectArrowThumbUp::Create
//--- example for CChartObjectArrowLeftPrice::Create
//--- example for CChartObjectArrowRightPrice::Create
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
//--- for example, take CChartObjectArrowCheck
   CChartObjectArrowCheck arrow;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))
```

```
      {
       //--- arrow create error
       printf("Arrow create: Error %d!",GetLastError());
       //---
       return;
      }
//--- use arrow
//--- . . .
  }
```

ArrowCode

Prohibits code changes for "Arrow".

```
bool  ArrowCode(
   char  code      // code value
   )
```

Parameters

code

[in] any value

Returned value

Always false.

Example:

```
//--- example for CChartObjectArrowCheck::ArrowCode
//--- example for CChartObjectArrowDown::ArrowCode
//--- example for CChartObjectArrowUp::ArrowCode
//--- example for CChartObjectArrowStop::ArrowCode
//--- example for CChartObjectArrowThumbDown::ArrowCode
//--- example for CChartObjectArrowThumbUp::ArrowCode
//--- example for CChartObjectArrowLeftPrice::ArrowCode
//--- example for CChartObjectArrowRightPrice::ArrowCode
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
//--- for example, take CChartObjectArrowCheck
   CChartObjectArrowCheck arrow;
//--- set object parameters
   double price=SymbolInfoDouble(Symbol(),SYMBOL_BID);
   if(!arrow.Create(0,"ArrowCheck",0,TimeCurrent(),price))
     {
      //--- arrow create error
      printf("Arrow create: Error %d!",GetLastError());
      //---
      return;
     }
//--- set code of arrow
   if(!arrow.ArrowCode(181))
     {
      //--- it is not error
      printf("Arrow code can not be changed");
     }
//--- use arrow
//--- . . .
  }
```

Type

Returns graphic object type identifier

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_ARROW_CHECK for CChartObjectArrowCheck, OBJ_ARROW_DOWN for CChartObjectArrowDown, OBJ_ARROW_UP for CChartObjectArrowUp, OBJ_ARROW_STOP for CChartObjectArrowStop, OBJ_ARROW_THUMB_DOWN for CChartObjectArrowThumbDown, OBJ_ARROW_THUMB_UP for CChartObjectArrowThumbUp, OBJ_ARROW_LEFT_PRICE for CChartObjectArrowLeftPrice, OBJ_ARROW_RIGHT_PRICE for CChartObjectArrowRightPrice).

Example:

```
//--- example for CChartObjectArrowCheck::Type
//--- example for CChartObjectArrowDown::Type
//--- example for CChartObjectArrowUp::Type
//--- example for CChartObjectArrowStop::Type
//--- example for CChartObjectArrowThumbDown::Type
//--- example for CChartObjectArrowThumbUp::Type
//--- example for CChartObjectArrowLeftPrice::Type
//--- example for CChartObjectArrowRightPrice::Type
#include <ChartObjects\ChartObjectsArrows.mqh>
//---
void OnStart()
  {
//--- for example, take CChartObjectArrowCheck
   CChartObjectArrowCheck arrow;
//--- get arrow type
   int type=arrow.Type();
  }
```

Object Controls

A group of graphic objects "Object Controls".

This section contains the technical details of working with a group of classes of graphical objects "Object Controls" and a description of the relevant components of the MQL5 Standard Library .

Class name	Object
CChartObjectText	Graphic object "Text"
CChartObjectLabel	Graphic object "Text Label"
CChartObjectEdit	Graphic object "Edit"
CChartObjectButton	Graphic object "Button"
CChartObjectSubChart	Graphic object "Chart"
CChartObjectBitmap	Graphic object "Bitmap"
CChartObjectBmpLabel	Graphic object "Bitmap Label"
CChartObjectRectLabel	Graphic object "Rectangle Label"

See also

Object types, Graphic objects

CChartObjectText

Class CChartObjectText is a class for simplified access to "Text" graphic object properties.

Description

Class CChartObjectText provides access to "Text" object properties.

Declaration

```
class CChartObjectText : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Text"
Properties	
Angle	Gets/Sets property "Angle"
Font	Gets/Sets property "Font"
FontSize	Gets/Sets property "FontSize"
Anchor	Gets/Sets property "Anchor"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Derived classes:

- CChartObjectLabel

See also

Object types, Object properties, Methods of binding sites, Graphic objects

Create

Creates graphic object "Text".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time,          // Time coordinate
   double    price          // Price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time

[in] Time coordinate of the anchor point.

price

[in] Price coordinate of the anchor point.

Returned value

true if successful, false if error.

Angle (Get Method)

Gets the value of "Angle" property.

```
double  Angle() const
```

Returned value

Value of "Angle" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Angle (Set Method)

Sets new value for "Angle" property.

```
bool  Angle(
   double  angle      // new angle
   )
```

Parameters

angle

[in] New value for "Angle" property.

Returned value

true if successful, false if property hasn't changed.

Font (Get Method)

Gets the value of "Font" property.

```
string  Font() const
```

Returned value

Value of "Font" property of the object, assigned to the class instance. If there is no object assigned, it returns "".

Font (Set Method)

Sets new value for "Font" property.

```
bool  Font(
   string  font      // new font
   )
```

Parameters

font

[in] New value for "Font" property.

Returned value

true if successful, false if property hasn't changed.

FontSize (Get Method)

Gets the value of "FontSize" property.

```
int  FontSize() const
```

Returned value

Value of "FontSize" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

FontSize (Set Method)

Sets new value for "FontSize" property.

```
bool  FontSize(
   int  size      // new font size
   )
```

Parameters

size

[in] New value for "Font" property.

Returned value

true if successful, false if property hasn't changed.

Anchor (Get Method)

Gets the value of "Anchor" property.

```
ENUM_ANCHOR_POINT  Anchor() const
```

Returned value

Value of "Anchor" property of the object, assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Anchor (Set Method)

Sets new value for "Anchor" property.

```
bool  Anchor(
   ENUM_ANCHOR_POINT  anchor      // new value
   )
```

Parameters

anchor

[in] New value for "Anchor" property.

Returned value

true if successful, false if property hasn't changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_TEXT for CChartObjectText).

CChartObjectLabel

Class CChartObjectLabel is a class for simplified access to "Label" graphic object properties.

Description

Class CChartObjectLabel provides access to "Label" object properties.

Declaration

```
class CChartObjectLabel : public CChartObjectText
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Label"
Properties	
X_Distance	Gets/Sets property "X_Distance"
Y_Distance	Gets/Sets property "Y_Distance"
X_Size	Gets/Sets property "X_Size"
Y_Size	Gets/Sets property "Y_Size"
Corner	Gets/Sets property "Corner"
Time	"Gag" for Time Coordinate change
Price	"Gag" for Price Coordinate change
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Derived classes:

- CChartObjectEdit

See also

Object types, Object properties, Chart angle, Methods of binding sites, Graphic objects

Create

Creates graphic object "Label".

```
bool  Create(
   long     chart_id,      // Chart identifier
   string   name,          // Object name
   int      window,        // Chart window
   int      X,             // X coordinate
   int      Y              // Y coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

X

[in] X coordinate.

Y

[in] Y coordinate.

Returned value

true if successful, false if error.

X_Distance (Get Method)

Gets the value of "X_Distance" property.

```
int  X_Distance() const
```

Returned value

Value of "X_Distance" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

X_Distance (Set Method)

Sets new value for "X_Distance" property.

```
bool  X_Distance(
   int  X      // new value
   )
```

Parameters

X

[in] New value for "X_Distance" property.

Returned value

true if successful, false if property hasn't changed.

Y_Distance (Get Method)

Gets the value of "Y_Distance" property.

```
int  Y_Distance() const
```

Returned value

Value of "Y_Distance" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Y_Distance (Set Method)

Sets new value for "Y_Distance" property.

```
bool  Y_Distance(
   int  Y      // new value
   )
```

Parameters

Y

[in] New value for "Y_Distance" property.

Returned value

true if successful, false if property hasn't changed.

X_Size

Gets the value of "X_Size" property.

```
int  X_Size() const
```

Returned value

Value of "X_Size" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Y_Size

Gets the value of "Y_Size" property.

```
int  Y_Size() const
```

Returned value

Value of "Y_Size" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Corner (Get Method)

Gets the value of "Corner" property.

```
ENUM_BASE_CORNER  Corner() const
```

Returned value

Value of "Corner" property of the object, assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Corner (Set Method)

Sets new value for "Corner" property.

```
bool  Corner(
   ENUM_BASE_CORNER  corner      // new value
   )
```

Parameters

corner

[in] New value for "Corner" property.

Returned value

true if successful, false if property hasn't changed.

Time

Prohibits changes of the time coordinate.

```
bool  Time(
   datetime  time      // any value
   )
```

Parameters

time

[in] Any value of datetime type.

Returned value

always false.

Price

Prohibits changes of the price coordinate.

```
bool  Price(
   double  price      // any value
   )
```

Parameters

price

[in] Any value of double type.

Returned value

always false.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_LABEL for CChartObjectLabel).

CChartObjectEdit

Class CChartObjectEdit is a class for simplified access to "Edit" graphic object properties.

Description

Class CChartObjectEdit provides access to "Edit" object properties.

Declaration

```
class CChartObjectEdit : public CChartObjectLabel
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Class Methods

Create	
Create	Creates "Edit" graphic object
Properties	
TextAlign	Gets/Sets "TextAlign" property
X_Size	Gets "X Size" property
Y_Size	Gets "Y Size" property
BackColor	Gets/Sets "Background Color" property
BorderColor	Gets/Sets "Border Color" property
Angle	Gets/Sets "Angle" property
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

Derived classes:

- CChartObjectButton

See also

Object types, Object properties, Chart angle, Methods of binding sites, Graphic objects

Create

Creates graphic object "Edit".

```
bool  Create(
   long     chart_id,      // Chart identifier
   string   name,          // Object name
   int      window,        // Chart window
   int      X,             // X coordinate
   int      Y,             // Y coordinate
   int      sizeX,         // X size
   int      sizeY          // Y size
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

X

[in] X coordinate.

Y

[in] Y coordinate.

sizeX

[in] X size.

sizeY

[in] Y size.

Returned value

true if successful, false if error.

X_Size

Sets new value for "X_Size" property.

```
bool  X_Size(
   int  size      // new size
   )
```

Parameters

size

[in] New value for "X_Size" property.

Returned value

true if successful, false if property hasn't changed.

Y_Size

Sets new value for "Y_Size" property.

```
bool  Y_Size(
   int  size      // new size
   )
```

Parameters

size

[in] New value for "Y_Size" property.

Returned value

true if successful, false if property hasn't changed.

BackColor (Get Method)

Gets the value of "BackColor" property.

```
color  BackColor() const
```

Returned value

Value of "BackColor" property of the object, assigned to the class instance. If there is no object assigned, it returns CLR_NONE.

BackColor (Set Method)

Sets new value for "BackColor" property.

```
bool  BackColor(
   color  new_color      // new background color
   )
```

Parameters

new_color

[in] New value for "BackColor" property.

Returned value

true if successful, false if property hasn't changed.

BorderColor (Get Method)

Gets the value of "Border Color" property.

```
color  BorderColor() const
```

Returned value

Value of "Border Color" property of the object, assigned to the class instance. If there is no object assigned, it returns CLR_NONE.

BorderColor (Set Method)

Sets new value for "Border Color" property.

```
bool  BorderColor(
   color  new_color      // new border color
   )
```

Parameters

new_color

[in] New value for "Border Color" property.

Returned value

true if successful, false if property hasn't changed.

Angle

Prohibits changes of the "Angle" property.

```
bool  Angle(
   double  angle      // any value
   )
```

Parameters

angle

[in] Any value of double type.

Returned value

always false.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_EDIT for CChartObjectEdit).

CChartObjectButton

Class CChartObjectButton is a class for simplified access to "Button" graphic object properties.

Description

Class CChartObjectButton provides access to "Button" object properties.

Declaration

```
class CChartObjectButton : public CChartObjectEdit
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Class Methods

Create	
Create	Inherited form class CChartObjectEdit
Properties	
State	Gets/Sets button state (Pressed/Depressed)
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Object properties, Chart angle, Methods of binding sites, Graphic objects

State (Get Method)

Gets the value of "State" property.

```
bool  State() const
```

Returned value

Value of "State" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

State (Set Method)

Sets new value for "State" property.

```
bool  State(
   bool  state      // new state value
   )
```

Parameters

X

[in] New value for "State" property.

Returned value

true if successful, false if property hasn't changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_BUTTON for CChartObjectButton).

CChartObjectSubChart

Class CChartObjectSubChart is a class for simplified access to "Chart" graphic object properties.

Description

Class CChartObjectSubChart provides access to "Chart" object properties.

Declaration

```
class CChartObjectSubChart : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectSubChart.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Chart"
Properties	
X_Distance	Gets/Sets property "X_Distance"
Y_Distance	Gets/Sets property "Y_Distance"
Corner	Gets/Sets property "Corner"
X_Size	Gets/Sets property "X_Size"
Y_Size	Gets/Sets property "Y_Size"
Symbol	Gets/Sets property "Symbol"
Period	Gets/Sets property "Period"
Scale	Gets/Sets property "Scale"
DateScale	Gets/Sets property "Show date scale"
PriceScale	Gets/Sets property "Show price scale"
Time	"Gag" for time coordinate change
Price	"Gag" for price coordinate change
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Object properties, Chart angle, Graphic objects

Create

Creates graphic object "SubChart".

```
bool  Create(
   long     chart_id,      // Chart identifier
   string   name,          // Object name
   int      window,        // Chart window
   int      X,             // X coordinate
   int      Y,             // Y coordinate
   int      sizeX,         // X size
   int      sizeY          // Y size
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

X

[in] X coordinate.

Y

[in] Y coordinate.

sizeX

[in] X size.

sizeY

[in] Y size.

Returned value

true if successful, false if error.

X_Distance (Get Method)

Gets the value of "X_Distance" property.

```
int  X_Distance() const
```

Returned value

Value of "X_Distance" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

X_Distance (Set Method)

Sets new value for "X_Distance" property.

```
bool  X_Distance(
   int  X      // new value
   )
```

Parameters

X

[in] New value for "X_Distance" property.

Returned value

true if successful, false if property hasn't changed.

Y_Distance (Get Method)

Gets the value of "Y_Distance" property.

```
int  Y_Distance() const
```

Returned value

Value of "Y_Distance" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Y_Distance (Set Method)

Sets new value for "Y_Distance" property.

```
bool  Y_Distance(
   int  Y      // new value
   )
```

Parameters

Y

[in] New value for "Y_Distance" property.

Returned value

true if successful, false if property hasn't changed.

Corner (Get Method)

Gets the value of "Corner" property.

```
ENUM_BASE_CORNER Corner() const
```

Returned value

Value of "Corner" property of the object, assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Corner (Set Method)

Sets new value for "Corner" property.

```
bool  Corner(
   ENUM_BASE_CORNER  corner      // new value
   )
```

Parameters

corner

[in] New value for "Corner" property.

Returned value

true if successful, false if property hasn't changed.

X_Size (Get Method)

Gets the value of "X_Size" property.

```
int  X_Size() const
```

Returned value

Value of "X_Size" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

X_Size (Set Method)

Sets new value for "X_Size" property.

```
bool  X_Size(
   int  X      // new value
   )
```

Parameters

X

[in] New value for "X_Size" property.

Returned value

true if successful, false if property hasn't changed.

Y_Size (Get Method)

Gets the value of "Y_Size" property.

```
int  Y_Size() const
```

Returned value

Value of "Y_Size" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Y_Size (Set Method)

Sets new value for "Y_Size" property.

```
bool  Y_Size(
   int  Y      // new value
   )
```

Parameters

Y

[in] New value for "Y_Size" property.

Returned value

true if successful, false if property hasn't changed.

Symbol (Get Method)

Gets the value of "Symbol" property.

```
string  Symbol() const
```

Returned value

Value of "Symbol" property of the object, assigned to the class instance. If there is no object assigned, it returns "".

Symbol (Set Method)

Sets new value for "Symbol" property.

```
bool  Symbol(
   string  symbol       // new symbol
   )
```

Parameters

symbol

[in] New value for "Symbol" property.

Returned value

true if successful, false if property hasn't changed.

Period (Get Method)

Gets the value of "Period" property.

```
int  Period() const
```

Returned value

Value of "Period" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Period (Set Method)

Sets new value for "Period" property.

```
bool  Period(
   int  period      // new period
   )
```

Parameters

period

[in] New value for "Period" property.

Returned value

true if successful, false if property hasn't changed.

Scale (Get Method)

Gets the value of "Scale" property.

```
double  Scale() const
```

Returned value

Value of "Scale" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

Scale (Set Method)

Sets new value for "Scale" property.

```
bool  Scale(
   double  scale      // new scale
   )
```

Parameters

scale

[in] New value for "Scale" property.

Returned value

true if successful, false if property hasn't changed.

DateScale (Get Method)

Gets the value of "DateScale" property.

```
bool  DateScale() const
```

Returned value

Value of "DateScale" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

DateScale (Set Method)

Sets new value for "DateScale" property.

```
bool  DateScale(
   bool  scale      // new value
   )
```

Parameters

scale

[in] New value for "DateScale" property.

Returned value

true if successful, false if property hasn't changed.

PriceScale (Get Method)

Gets the value of "PriceScale" property.

```
bool  PriceScale() const
```

Returned value

Value of "PriceScale" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

PriceScale (Set Method)

Sets new value for "PriceScale" property.

```
bool  PriceScale(
   bool  scale      // new value
   )
```

Parameters

scale

[in] New value for "PriceScale" property.

Returned value

true if successful, false if property hasn't changed.

Time

Prohibits changes of the time coordinate.

```
bool  Time(
   datetime  time      // any value
   )
```

Parameters

time

[in] Any value of datetime type.

Returned value

always false.

Price

Prohibits changes of the price coordinate.

```
bool  Price(
   double  price      // any value
   )
```

Parameters

price

[in] Any value of double type.

Returned value

always false.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_CHART for CChartObjectSubChart).

CChartObjectBitmap

Class CChartObjectBitmap is a class for simplified access to "Bitmap" graphic object properties.

Description

Class CChartObjectBitmap provides access to "Bitmap" object properties.

Declaration

```
class CChartObjectBitmap : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

Class Methods

Create	
Create	Creates graphic object "Bitmap"
Properties	
BmpFile	Gets/Sets property "BMP Filename"
X_Offset	Gets/Sets property "X_Offset"
Y_Offset	Gets/Sets property "Y_Offset"
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Object properties, Graphic objects

Create

Creates graphic object "Bitmap".

```
bool  Create(
   long      chart_id,      // Chart identifier
   string    name,          // Object name
   int       window,        // Chart window
   datetime  time,          // Time coordinate
   double    price          // Price coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] An unique name of the object to create.

window

[in] Chart window number (0 - base window).

time

[in] Time coordinate of the anchor point.

price

[in] Price coordinate of the anchor point.

Returned value

true if successful, false if error.

BmpFile (Get Method)

Gets the value of "BmpFile" property.

```
string  BmpFile() const
```

Returned value

Value of "BmpFile" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

BmpFile (Set Method)

Sets new value for "BmpFile" property.

```
bool  BmpFile(
   string  name      // new file name
   )
```

Parameters

X

[in] New value for "BmpFile" property.

Returned value

true if successful, false if property hasn't changed.

X_Offset (Get Method)

Gets the value of "X_Offset" property (the X coordinate of the upper left corner of the rectangular visible area in the graphical objects).

```
int  X_Offset() const
```

Returned value

Value of "X_Offset" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

X_Offset (Set Method)

Sets new value for "X_Offset" property (the X coordinate of the upper left corner of the rectangular visible area in the graphical objects). The value is set in pixels relative to the upper left corner of the original image.

```
bool  X_Offset(
   int  X      // new value
   )
```

Parameters

X

[in] New value for "X_Offset" property.

Returned value

true if successful, false if property hasn't changed.

Y_Offset (Get Method)

Gets the value of "Y_Offset" property (the Y coordinate of the upper left corner of the rectangular visible area in the graphical objects).

```
int  Y_Offset() const
```

Returned value

Value of "Y_Offset" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Y_Offset (Set Method)

Sets new value for "X_Offset" property (the Y coordinate of the upper left corner of the rectangular visible area in the graphical objects). The value is set in pixels relative to the upper left corner of the original image.

```
bool  Y_Offset(
   int  Y      // new value
   )
```

Parameters

Y

[in] New value for "Y_Offset" property.

Returned value

true if successful, false if property hasn't changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_BITMAP for CChartObjectBitmap).

CChartObjectBmpLabel

Class CChartObjectBmpLabel is a class for simplified access to "Bitmap Label" graphic object properties.

Description

Class CChartObjectBmpLabel provides access to "Bitmap Label" object properties.

Declaration

```
class CChartObjectBmpLabel : public CChartObject
```

Title

```
#include <ChartObjects\ChartObjectsBmpControls.mqh>
```

Class Methods

Create	
Create	Creates graphic object "BmpLabel"
Properties	
X_Distance	Gets/Sets property "X_Distance"
Y_Distance	Gets/Sets property "Y_Distance"
X_Offset	Gets/Sets property "X_Offset"
Y_Offset	Gets/Sets property "Y_Offset"
Corner	Gets/Sets property "Corner"
X_Size	Gets/Sets property "X_Size"
Y_Size	Gets/Sets property "Y_Size"
BmpFileOn	Gets/Sets property "BmpFileOn" for button pressed state (On)
BmpFileOff	Gets/Sets property "BmpFileOff" for button depressed state (Off)
State	Gets/Sets property "Button State" (Pressed/ Depressed)
Time	"Gag" for time coordinate change
Price	"Gag" for price coordinate change
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file

virtual Type	Virtual method of identification

See also

Object types, Object properties, Chart angle, Graphic objects

Create

Creates graphic object "BmpLabel".

```
bool  Create(
   long     chart_id,      // Chart identifier
   string   name,          // Object name
   int      window,        // Chart window
   int      X,             // X coordinate
   int      Y              // Y coordinate
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - base window).

X

[in] X coordinate.

Y

[in] Y coordinate.

Returned value

true if successful, false if error.

X_Distance (Get Method)

Gets the value of "X_Distance" property.

```
int  X_Distance() const
```

Returned value

Value of "X_Distance" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

X_Distance (Set Method)

Sets new value for "X_Distance" property.

```
bool  X_Distance(
   int  X      // new value
   )
```

Parameters

X

[in] New value for "X_Distance" property.

Returned value

true if successful, false if property hasn't changed.

Y_Distance (Get Method)

Gets the value of "Y_Distance" property.

```
int  Y_Distance() const
```

Returned value

Value of "Y_Distance" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Y_Distance (Set Method)

Sets new value for "Y_Distance" property.

```
bool  Y_Distance(
   int  Y      // new value
   )
```

Parameters

Y

[in] New value for "Y_Distance" property.

Returned value

true if successful, false if property hasn't changed.

X_Offset (Get Method)

Gets the value of "X_Offset" property (the X coordinate of the upper left corner of the rectangular visible area in the graphical objects).

```
int  X_Offset() const
```

Returned value

Value of "X_Offset" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

X_Offset (Set Method)

Sets new value for "X_Offset" property (the X coordinate of the upper left corner of the rectangular visible area in the graphical objects). The value is set in pixels relative to the upper left corner of the original image.

```
bool  X_Offset(
   int  X      // new value
   )
```

Parameters

X

[in] New value for "X_Offset" property.

Returned value

true if successful, false if property hasn't changed.

Y_Offset (Get Method)

Gets the value of "Y_Offset" property (the Y coordinate of the upper left corner of the rectangular visible area in the graphical objects).

```
int  Y_Offset() const
```

Returned value

Value of "Y_Offset" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Y_Offset (Set Method)

Sets new value for "X_Offset" property (the Y coordinate of the upper left corner of the rectangular visible area in the graphical objects). The value is set in pixels relative to the upper left corner of the original image.

```
bool  Y_Offset(
   int  Y      // new value
   )
```

Parameters

Y

[in] New value for "Y_Offset" property.

Returned value

true if successful, false if property hasn't changed.

Corner (Get Method)

Gets the value of "Corner" property.

```
ENUM_BASE_CORNER Corner() const
```

Returned value

Value of "Corner" property of the object, assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Corner (Set Method)

Sets new value for "Corner" property.

```
bool  Corner(
   ENUM_BASE_CORNER  corner      // new value
   )
```

Parameters

corner

[in] New value for "Corner" property.

Returned value

true if successful, false if property hasn't changed.

X_Size

Gets the value of "X_Size" property.

```
int  X_Size() const
```

Returned value

Value of "X_Size" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Y_Size

Gets the value of "Y_Size" property.

```
int  Y_Size() const
```

Returned value

Value of "Y_Size" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

BmpFileOn (Get Method)

Gets the value of "BmpFileOn" property.

```
string  BmpFileOn() const
```

Returned value

Value of "BmpFileOn" property of the object, assigned to the class instance. If there is no object assigned, it returns "".

BmpFileOn (Set Method)

Sets new value for "BmpFileOn" property.

```
bool  BmpFileOn(
   string  name      // file name
   )
```

Parameters

name

[in] New value for "BmpFileOn" property.

Returned value

true if successful, false if property hasn't changed.

BmpFileOff (Get Method)

Gets the value of "BmpFileOff" property.

```
string  BmpFileOff() const
```

Returned value

Value of "BmpFileOff" property of the object, assigned to the class instance. If there is no object assigned, it returns "".

BmpFileOff (Set Method)

Sets new value for "BmpFileOff" property.

```
bool  BmpFileOff(
   string  name      // file name
   )
```

Parameters

name

[in] New value for "BmpFileOff" property.

Returned value

true if successful, false if property hasn't changed.

State (Get Method)

Gets the value of "State" property.

```
bool  State() const
```

Returned value

Value of "State" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

State (Set Method)

Sets new value for "State" property.

```
bool  State(
   bool  state      // new state value
   )
```

Parameters

X

[in] New value for "State" property.

Returned value

true if successful, false if property hasn't changed.

Time

Prohibits changes of the time coordinate.

```
bool  Time(
   datetime  time      // any value
   )
```

Parameters

time

[in] Any value of datetime type.

Returned value

always false.

Price

Prohibits changes of the price coordinate.

```
bool  Price(
   double  price      // any value
   )
```

Parameters

price

[in] Any value of double type.

Returned value

always false.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_BITMAP_LABEL for CChartObjectBmpLabel).

CChartObjectRectLabel

Class CChartObjectRectLabel is a class for simplified access to "Rectangle Label" graphic object properties.

Description

Class CChartObjectRectLabel provides access to "Rectangle Label" object properties.

Declaration

```
class CChartObjectRectLabel : public CChartObjectLabel
```

Title

```
#include <ChartObjects\ChartObjectsTxtControls.mqh>
```

Class Methods

Create	
Create	Creates "RectLabel" graphic object
Properties	
X_Size	Sets the horizontal size
Y_Size	Sets the vertical size
BackColor	Gets/Sets the background color
Angle	A gag method
BorderType	Gets/Sets type of the border
Input/output	
virtual Save	Virtual method for writing to file
virtual Load	Virtual method for reading from file
virtual Type	Virtual method of identification

See also

Object types, Object properties, Graphic objects

Create

Creates the "CChartObjectRectLabel" graphic object.

```
bool  Create(
   long     chart_id,      // Chart ID
   string   name,          // Object name
   int      window,        // Chart window
   int      X,             // X coordinate
   int      Y,             // Y coordinate
   int      sizeX,         // Horizontal size
   int      sizeY          // Vertical size
   )
```

Parameters

chart_id

[in] Chart identifier (0 - current chart).

name

[in] A unique name of the object to create.

window

[in] Chart window number (0 - base window).

X

[in] X coordinate.

Y

[in] Y coordinate.

sizeX

[in] Horizontal size.

sizeY

[in] Vertical size.

Returned value

true if successful, false if error.

X_Size

Sets the value of "X_Size" property.

```
bool  X_Size(
   int  size      // Horizontal size
   )
```

Parameters

size

[in] New horizontal size.

Returned value

true if successful, false if the property hasn't been changed.

Note

To get the values of "X_Size" and "Y_Size" properties use the X_Size and Y_Size methods of the parent CChartObjectLabel class.

Y_Size

Sets the value of "Y_Size" property.

```
bool  Y_Size(
   int  size      // Vertical size
   )
```

Parameters

size

[in] New vertical size.

Returned value

true if successful, false if the property hasn't been changed.

Note

To get the values of "X_Size" and "Y_Size" properties use the X_Size and Y_Size methods of the parent CChartObjectLabel class.

BackColor

Gets the background color.

```
color  BackColor() const
```

Returned value

Background color of the object, assigned to the class instance. If there is no object assigned, it returns 0.

BackColor

Sets the background color.

```
bool  BackColor(
   color  new_color      // New color
   )
```

Parameters

new_color

[in] New background color.

Returned value

true if successful, false if property hasn't been changed.

Angle

A gag method.

```
bool  Angle(
   double  angle      // any value
   )
```

Parameters

angle

[in] Any value of double type.

Returned value

Always false.

BorderType

Gets border type.

```
int  BorderType() const
```

Returned value

Border type of the object, assigned to the class instance. If there is no object assigned, it returns 0.

BorderType

Sets border type.

```
bool  BorderType(
   int  type      // Border type
   )
```

Parameters

type

[in] New border type.

Returned value

true if successful, false if property hasn't been changed.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen function.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (OBJ_RECTANGLE_LABEL for CChartObjectRectangleLabel).

CCanvas

CCanvas is a class for simplified creation of custom images.

Description

CCanvas provides creation of a graphical resource (with or without binding to a chart object) and drawing graphic primitives.

Declaration

```
class CCanvas
```

Title

```
#include <Canvas\Canvas.mqh>
```

Class Methods by Groups

Creating	
Create	Creates a graphical resource without binding to a chart object
CreateBitmap	Creates a graphical resource bound to a chart object
CreateBitmapLabel	Creates a graphical resource bound to a chart object
Destroy	Destroys a graphical resource
Properties	
ChartObjectName	Receives the name of a bound chart object
ResourceName	Receives the name of a graphical resource
Width	Receives the width of a graphical resource
Height	Receives the height of a graphical resource
LineStyleSet	Sets the line style
Updating an object on the screen	
Update	Displays changes on the screen
Resize	Resizes a graphical resource
Erasing/Filling with color	
Erase	Erases or fills with the specified color
Data access	
PixelGet	Receives color of the dot with the specified coordinates

PixelSet	Sets color of the dot with the specified coordinates
Drawing primitives	
LineVertical	Draws a vertical line
LineHorizontal	Draws a horizontal line
Line	Draws a freehand line
Polyline	Draws a polyline
Polygon	Draws a polygon
Rectangle	Draws a rectangle
Circle	Draws a circle
Triangle	Draws a triangle
Drawing filled primitives	
FillRectangle	Draws a filled rectangle
FillCircle	Draws a filled circle
FillTriangle	Draws a filled triangle
Fill	Fills an area
Drawing primitives with antialiasing	
PixelSetAA	Draws a pixel
LineAA	Draws a line
PolylineAA	Draws a polyline
PolygonAA	Draws a polygon
TriangleAA	Draws a triangle
CircleAA	Draws a circle
Text	
FontSet	Sets font parameters
FontNameSet	Sets font name
FontSizeSet	Sets font size
FontFlagsSet	Sets font flags
FontAngleSet	Sets font slope angle
FontGet	Receives font parameters
FontNameGet	Receives font name
FontSizeGet	Receives font size
FontFlagsGet	Receives font flags

FontAngleGet	Receives font slope angle
TextOut	Displays text
TextWidth	Receives the text width
TextHeight	Receives the text height
TextSize	Receives the text size
Transparency	
TransparentLevelSet	Sets transparency level
Input/output	
LoadFromFile	Reads an image from a BMP file

ChartObjectName

Receives the name of a bound chart object.

```
string  ChartObjectName();
```

Returned value

the name of a bound chart object

Circle

Draws a circle

```
void  Circle(
   int          x,        // X coordinate
   int          y,        // Y coordinate
   int          r,        // radius
   const uint   clr       // color
   );
```

Parameters

x

[in] X coordinate of the center of the circle.

y

[in] Y coordinate of the center of the circle.

r

[in] Circle radius.

clr

[in] Color in ARGB format.

CircleAA

Draws a circle using antialiasing algorithm

```
void  CircleAA(
   const int      x,        // X coordinate
   const int      y,        // Y coordinate
   const double   r,        // radius
   const uint     clr       // color
   );
```

Parameters

x

[in] X coordinate of the center of the circle.

y

[in] Y coordinate of the center of the circle.

r

[in] Circle radius.

clr

[in] Color in ARGB format.

Create

Creates a graphical resource without binding to a chart object.

```
virtual bool  Create(
   const string       name,                              // name
   const int          width,                             // width
   const int          height,                            // height
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA   // format
   );
```

Parameters

name

[in] Basis for a graphical resource name. A resource name is generated during the creation by adding a pseudorandom string.

width

[in] Width (size along X axis) in pixels.

height

[in] Height (size along Y axis) in pixels.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color processing method. See ResourceCreate() function description to learn more about color processing methods.

Returned value

true - if successful, otherwise - false

CreateBitmap

Creates a graphical resource bound to a chart object.

1. Creates a graphical resource in the main window of the current chart.

```
bool  CreateBitmap(
   const string       name,                                  // name
   const datetime     time,                                  // time
   const double       price,                                 // price
   const int          width,                                 // width
   const int          height,                                // height
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // format
   );
```

2. Creates a graphical resource using a chart ID and a subwindow number.

```
bool  CreateBitmap(
   const long         chart_id,                              // chart ID
   const int          subwin,                                // subwindow number
   const string       name,                                  // name
   const datetime     time,                                  // time
   const double       price,                                 // price
   const int          width,                                 // width
   const int          height,                                // height
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA       // format
   );
```

Parameters

chart_id

[in] Chart ID for creating an object.

subwin

[in] Chart subwindow number for creating an object.

name

[in] Chart object name and a basis for a graphical resource name.

time

[in] Chart object anchor point time coordinate.

price

[in] Chart object anchor point price coordinate.

width

[in] Graphical resource width (size along X axis) in pixels.

height

[in] Graphical resource height (size along Y axis) in pixels.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color processing method. See ResourceCreate() function description to learn more about color processing methods.

Returned value

true - if successful, otherwise - false

Note

If the first function version is used, the object is created in the main window of the current chart.

Object size coincides with the size of a graphical resource.

CreateBitmapLabel

Creates a graphical resource bound to a chart object.

1. Creates a graphical resource in the main window of the current chart.

```
bool  CreateBitmapLabel(
   const string       name,                              // name
   const int          x,                                 // X coordinate
   const int          y,                                 // Y coordinate
   const int          width,                             // width
   const int          height,                            // height
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA   // format
   );
```

2. Creates a graphical resource using a chart ID and a subwindow number.

```
bool  CreateBitmapLabel(
   const long         chart_id,                          // chart ID
   const int          subwin,                            // subwindow number
   const string       name,                              // name
   const int          x,                                 // X coordinate
   const int          y,                                 // Y coordinate
   const int          width,                             // width
   const int          height,                            // height
   ENUM_COLOR_FORMAT  clrfmt=COLOR_FORMAT_XRGB_NOALPHA   // format
   );
```

Parameters

chart_id

[in] Chart ID for creating an object.

subwin

[in] Chart subwindow number for creating an object.

name

[in] Chart object name and a basis for a graphical resource name.

x

[in] Chart object anchor point X coordinate.

y

[in] Chart object anchor point Y coordinate.

width

[in] Graphical resource width (size along X axis) in pixels.

height

[in] Graphical resource height (size along Y axis) in pixels.

clrfmt=COLOR_FORMAT_XRGB_NOALPHA

[in] Color processing method. See ResourceCreate() function description to learn more about color processing methods.

Returned value

true - if successful, otherwise - false

Note

If the first function version is used, the object is created in the main window of the current chart.

Object size coincides with the size of a graphical resource.

Destroy

Destroys a graphical resource.

```
void  Destroy();
```

Note

If a graphical resource has been bound to a chart object, the latter is deleted.

Erase

Erases or fills with the specified color.

```
void  Erase(
   const uint  clr=0      // color
   );
```

Parameters

clr=0

[in] Color in ARGB format.

Fill

Fills an area.

```
void  Fill(
   int          x,        // X coordinate
   int          y,        // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x

[in] X coordinate of filling starting point.

y

[in] Y coordinate of filling starting point.

clr

[in] Color in ARGB format.

FillCircle

Draws a filled circle.

```
void  FillCircle(
   int          x,        // X coordinate
   int          y,        // Y coordinate
   int          r,        // radius
   const uint   clr       // color
   );
```

Parameters

x

[in] X coordinate of a filled circle center.

y

[in] Y coordinate of a filled circle center.

r

[in] Filled circle radius.

clr

[in] Color in ARGB format.

FillRectangle

Draws a filled rectangle.

```
void  FillRectangle(
   int          x1,      // X coordinate
   int          y1,      // Y coordinate
   int          x2,      // X coordinate
   int          y2,      // Y coordinate
   const uint   clr      // color
   );
```

Parameters

x1

[in] X coordinate of the first point forming a rectangle.

y1

[in] Y coordinate of the first point forming a rectangle.

x2

[in] X coordinate of the second point forming a rectangle.

y2

[in] Y coordinate of the second point forming a rectangle.

clr

[in] Color in ARGB format.

FillTriangle

Draws a filled triangle.

```
void  FillTriangle(
   int          x1,      // X coordinate
   int          y1,      // Y coordinate
   int          x2,      // X coordinate
   int          y2,      // Y coordinate
   int          x3,      // X coordinate
   int          y3,      // Y coordinate
   const uint   clr      // color
   );
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

FontAngleGet

Receives font slope angle.

```
uint  FontAngleGet();
```

Returned value

font slope angle

FontAngleSet

Sets font slope angle.

```
bool  FontAngleSet(
   uint  angle      // angle
   );
```

Parameters

angle

[in] Font slope angle in tenths of a degree.

Returned value

true - if successful, otherwise - false

FontFlagsGet

Receives font flags.

```
uint  FontFlagsGet();
```

Returned value

font flags

FontFlagsSet

Sets font flags.

```
bool  FontFlagsSet(
   uint  flags      // flags
   );
```

Parameters

flags

[in] Font creation flags. See TextSetFont() function description to learn more about the flags.

Returned value

true - if successful, otherwise - false

FontGet

Receives the current font parameters.

```
void  FontGet(
   string&  name,      // name
   int&     size,      // size
   uint&    flags,     // flags
   uint&    angle      // slope angle
   );
```

Parameters

name

[out] Reference to the variable for returning a font name.

size

[out] Reference to the variable for returning a font size.

flags

[out] Reference to the variable for returning font flags.

angle

[out] Reference to the variable for returning a font slope angle.

FontNameGet

Receives font name.

```
string  FontNameGet();
```

Returned value

font name

FontNameSet

Sets font name.

```
bool  FontNameSet(
   string  name      // name
   );
```

Parameters

name

[in] Font name. For example, "Arial".

Returned value

true - if successful, otherwise - false

FontSet

Sets the current font.

```
bool  FontSet(
   const string  name,        // name
   const int     size,        // size
   const uint    flags=0,     // flags
   const uint    angle=0      // angle
   );
```

Parameters

name

[in] Font name. For example, "Arial".

size

[in] Font size. See TextSetFont() function description to learn more about setting a size.

flags=0

[in] Font creation flags. See TextSetFont() function description to learn more about the flags.

angle=0

[in] Font slope angle in tenths of a degree.

Returned value

true - if successful, otherwise - false

FontSizeGet

Receives font size.

```
int  FontSizeGet();
```

Returned value

font size

FontSizeSet

Sets font size.

```
bool  FontSizeSet(
   int  size      // size
   );
```

Parameters

size

[in] Font size. See TextSetFont() function description to learn more about setting a size.

Returned value

true - if successful, otherwise - false

Height

Receives the height of a graphical resource.

```
int  Height();
```

Returned value

height of a graphical resource

Line

Draws a segment of a freehand line.

```
void  Line(
   int          x1,      // X coordinate
   int          y1,      // Y coordinate
   int          x2,      // X coordinate
   int          y2,      // Y coordinate
   const uint   clr      // color
   );
```

Parameters

x1

[in] X coordinate of the segment's first point.

y1

[in] Y coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

LineAA

Draws a segment of a freehand line using antialiasing algorithm.

```
void  LineAA(
   const int   x1,                  // X coordinate
   const int   y1,                  // Y coordinate
   const int   x2,                  // X coordinate
   const int   y2,                  // Y coordinate
   const uint  clr,                 // color
   const uint  style=UINT_MAX       // line style
   );
```

Parameters

x1

[in] X coordinate of the segment's first point.

y1

[in] Y coordinate of the segment's first point.

x2

[in] X coordinate of the segment's second point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

LineHorizontal

Draws a segment of a horizontal line.

```
void  LineHorizontal(
   int          x1,       // X coordinate
   int          x2,       // X coordinate
   int          y,        // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x1

[in] Segment's X coordinate.

x2

[in] X coordinate of the segment's first point.

y

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

LineStyleSet

Sets the line style.

```
void  LineStyleSet(
   const uint  style      // style
   );
```

Parameters

style

[in] Line style.

Note

The input parameter can have any of ENUM_LINE_STYLE enumeration values. Besides, it is possible to create a custom line drawing style.

LineVertical

Draws a segment of a vertical line.

```
void  LineVertical(
   int          x,        // X coordinate
   int          y1,       // Y coordinate
   int          y2,       // Y coordinate
   const uint   clr       // color
   );
```

Parameters

x

[in] Segment's X coordinate.

y1

[in] Y coordinate of the segment's first point.

y2

[in] Y coordinate of the segment's second point.

clr

[in] Color in ARGB format.

LoadFromFile

Reads an image from a BMP file.

```
bool  LoadFromFile(
   const string  filename      // file name
   );
```

Parameters

filename

[in] File name (including "BMP" extension).

Returned value

true - if successful, otherwise - false

PixelGet

Receives color of the point with the specified coordinates.

```
uint  PixelGet(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   );
```

Parameters

x

[in] Point's X coordinate.

y

[in] Point's Y coordinate.

Returned value

Point color in ARGB format.

PixelSet

Sets color of the point with the specified coordinates.

```
void  PixelSet(
   const int   x,       // X coordinate
   const int   y,       // Y coordinate
   const uint  clr      // color
   );
```

Parameters

x

[in] Point's X coordinate.

y

[in] Point's Y coordinate.

clr

[in] Color in ARGB format.

PixelSetAA

Draws a point using antialiasing algorithm.

```
void  PixelSetAA(
   const double  x,        // X coordinate
   const double  y,        // Y coordinate
   const uint    clr       // color
   );
```

Parameters

x

[in] Point's X coordinate.

y

[in] Point's Y coordinate.

clr

[in] Color in ARGB format.

Polygon

Draws a polygon.

```
void  Polygon(
   int&         x[],     // array of X coordinates
   int&         y[],     // array of Y coordinates
   const uint   clr      // color
   );
```

Parameters

x[]

[in] Array of X coordinates of a polygon points.

y[]

[in] Array of Y coordinates of a polygon points.

clr

[in] Color in ARGB format.

PolygonAA

Draws a polygon using antialiasing algorithm.

```
void  PolygonAA(
   int&         x[],                // array of X coordinates
   int&         y[],                // array of Y coordinates
   const uint   clr,                // color
   const uint   style=UINT_MAX      // line style
   );
```

Parameters

x[]

[in] Array of X coordinates of a polygon points.

y[]

[in] Array of Y coordinates of a polygon points.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

Polyline

Draws a polyline.

```
void  Polyline(
   int&         x[],     // array of X coordinates
   int&         y[],     // array of Y coordinates
   const uint   clr      // color
   );
```

Parameters

x[]

[in] Array of X coordinates of a polyline.

y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

PolylineAA

Draws a polyline using antialiasing algorithm.

```
void  PolylineAA(
   int&         x[],                // array of X coordinates
   int&         y[],                // array of Y coordinates
   const uint   clr,                // color
   const uint   style=UINT_MAX      // line style
   );
```

Parameters

x[]

[in] Array of X coordinates of a polyline.

y[]

[in] Array of Y coordinates of a polyline.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

Rectangle

Draws a rectangle using two points.

```
void  Rectangle(
   int          x1,      // X coordinate
   int          y1,      // Y coordinate
   int          x2,      // X coordinate
   int          y2,      // Y coordinate
   const uint   clr      // color
   );
```

Parameters

x1

[in] X coordinate of the first point forming a rectangle.

y1

[in] Y coordinate of the first point forming a rectangle.

x2

[in] X coordinate of the second point forming a rectangle.

y2

[in] Y coordinate of the second point forming a rectangle.

clr

[in] Color in ARGB format.

Resize

Resizes a graphical resource.

```
bool  Resize(
   const int  width,       // width
   const int  height       // height
   );
```

Parameters

width

[in] New width of a graphical resource.

height

[in] New height of a graphical resource.

Returned value

true - if successful, otherwise - false

Note

When resizing, the previous image is not saved.

ResourceName

Receives the name of a graphical resource.

```
string  ResourceName();
```

Returned value

name of a graphical resource

TextHeight

Receives the text height.

```
int  TextHeight(
   const string  text      // text
   );
```

Parameters

text

[in] Text for measuring.

Returned value

text height in pixels

Note

The current font is used for measuring the text.

TextOut

Displays text.

```
void  TextOut(
   int          x,               // X coordinate
   int          y,               // Y coordinate
   string       text,            // text
   const uint   clr,             // color
   uint         alignment=0      // alignment
   );
```

Parameters

x

[in] Text anchor's X coordinate.

y

[in] Text anchor's Y coordinate.

text

[in] Text to be displayed.

clr

[in] Color in ARGB format.

alignment=0

[in] Text anchoring method. See TextOut() function description to learn more about anchoring methods.

Note

The current font is used to display the text.

TextSize

Receives the text size.

```
void  TextSize(
   const string  text,        // text
   int&          width,       // width
   int&          height       // height
   );
```

Parameters

text

[in] Text for measuring.

width

[out] Reference to the variable for returning a text width.

height

[out] Reference to the variable for returning a text height.

Note

The current font is used to measure the text.

TextWidth

Receives the text width.

```
int  TextWidth(
   const string  text      // text
   );
```

Parameters

text

[in] Text for measuring.

Returned value

text height in pixels

Note

The current font is used to measure the text.

TransparentLevelSet

Sets transparency level.

```
void  TransparentLevelSet(
   const uchar  value      // value
   );
```

Parameters

value

[in] New value of the transparency level.

Note

0 stands for full transparency, while 255 - for full opacity.

Setting of a transparency level affects all that was previously drawn. The specified transparency level does not affect further constructions.

Triangle

Draws a triangle.

```
void  Triangle(
   int          x1,      // X coordinate
   int          y1,      // Y coordinate
   int          x2,      // X coordinate
   int          y2,      // Y coordinate
   int          x3,      // X coordinate
   int          y3,      // Y coordinate
   const uint   clr      // color
   );
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

TriangleAA

Draws a triangle using antialiasing algorithm.

```
void  TriangleAA(
   const int   x1,                  // X coordinate
   const int   y1,                  // Y coordinate
   const int   x2,                  // X coordinate
   const int   y2,                  // Y coordinate
   const int   x3,                  // X coordinate
   const int   y3,                  // Y coordinate
   const uint  clr,                 // color
   const uint  style=UINT_MAX       // line style
   );
```

Parameters

x1

[in] X coordinate of the triangle's first corner.

y1

[in] Y coordinate of the triangle's first corner.

x2

[in] X coordinate of the triangle's second corner.

y2

[in] Y coordinate of the triangle's second corner.

x3

[in] X coordinate of the triangle's third corner.

y3

[in] Y coordinate of the triangle's third corner.

clr

[in] Color in ARGB format.

style=UINT_MAX

[in] Line style is one of ENUM_LINE_STYLE enumeration's values or a custom value.

Update

Displays changes on the screen.

```
void  Update(
   const bool  redraw=true      // flag
   );
```

Parameters

redraw=true

Flag of a chart redrawing necessity.

Width

Receives the width of a graphical resource.

```
int  Width();
```

Returned value

graphical resource width

CChart

Class CChart is a class for simplified access to "Chart" graphic object properties.

Description

Class CChart provides access to "Chart" object properties.

Declaration

```
class CChart : public CObject
```

Title

```
<Charts\Chart.mqh>
```

Class Methods

Access to protected data	
ChartID	Gets identifier of the chart
General properties	
Mode	Gets/Sets the value of "Mode" property (bar, candle or line)
Foreground	Gets/Sets the value of "Foreground" property
Shift	Gets/Sets the value of "Shift" property
ShiftSize	Gets/Sets the value of "ShiftSize" property (in percents)
AutoScroll	Gets/Sets the value of "AutoScroll" property
Scale	Gets/Sets the value of "Scale" property
ScaleFix	Gets/Sets the value of "ScaleFix" property (fixed chart scale or not)
ScaleFix_11	Gets/Sets the value of "ScaleFix_11" property (chart scale is 1:1, or not)
FixedMax	Gets/Sets the value of "FixedMax" property (fixed maximal price)
FixedMin	Gets/Sets the value of "FixedMin" property (fixed minimal price)
ScalePPB	Gets/Sets the value of "ScalePPB" property (scale is "point per bar" or not)
PointsPerBar	Gets/Sets the value of "PointsPerBar" property (in points per bar)
Show properties	

ShowOHLC	Gets/Sets the value of "ShowOHLC" property
ShowLineBid	Gets/Sets the value of "ShowLineBid" property
ShowLineAsk	Gets/Sets the value of "ShowLineAsk" property
ShowLastLine	Gets/Sets the value of "ShowLastLine" property
ShowPeriodSep	Gets/Sets the value of "ShowPeriodSep" property (show period separators)
ShowGrid	Gets/Sets the value of "ShowGrid" property
ShowVolumes	Gets/Sets the value of "ColorVolumes" property (color for volumes and levels of opened positions)
ShowObjectDescr	Gets/Sets the value of "ShowObjectDescr" property (show description for graphic objects)
ShowDateScale	Sets the value of "ShowDateScale" property (date scale of the chart)
ShowPriceScale	Sets the value of "ShowPriceScale" property (price scale of the chart)
Colors properties	
ColorBackground	Gets/Sets the value of "ColorBackground" property (background color of the chart)
ColorForeground	Gets/Sets the value of "ColorForeground" property (color of axes, scale and OHLC strings of the chart)
ColorGrid	Gets/Sets the value of "ColorGrid" property (color of the grid)
ColorBarUp	Gets/Sets the value of "ColorBarUp" property (color for bull bars, its shadow and candle body outlines)
ColorBarDown	Gets/Sets the value of "ColorBarDown" property (color for bear bars, its shadow and candle body outlines)
ColorCandleBull	Gets/Sets the value of "ColorCandleBull" property (body color of the bull candle)
ColorCandleBear	Gets/Sets the value of "ColorCandleBear" property (body color of the bear candle)
ColorChartLine	Gets/Sets the value of "ColorChartLine" property (color for line chart and Doji candles)
ColorVolumes	Gets/Sets the value of "ColorVolumes" property (color for volumes and levels of opened positions)

ColorLineBid	Gets/Sets the value of "ColorLineBid" property (color of Bid line)
ColorLineAsk	Gets/Sets the value of "ColorLineAsk" property (color of Ask line)
ColorLineLast	Gets/Sets the value of "ColorLineLast" property (color of the last deal price line)
ColorStopLevels	Gets/Sets the value of "ColorStopLevels" property (color of the SL and TP levels)
Read only properties	
VisibleBars	Gets total number of visible chart bars
WindowsTotal	Gets total number of chart windows, including the chart indicator subwindows
WindowIsVisible	Gets visibility flag of the specified chart subwindow
WindowHandle	Gets window handle of the chart (HWND)
FirstVisibleBar	Gets the number of the first visible bar of the chart
WidthInBars	Gets window width in bars.
WidthInPixels	Gets subwindow width in pixels.
HeightInPixels	Gets subwindow height in pixels.
PriceMin	Gets minimal price of the specified subwindow
PriceMax	Gets maximal price of the specified subwindow
Properties	
Attach	Assigns the current chart to the class instance
FirstChart	Assigns the first chart of the client terminal to the class instance
NextChart	Assigns the next chart of the client terminal to the class instance
Open	Opens chart with specified parameters and assign it to the class instance
Detach	Detaches chart from the class instance
Close	Closes chart, assigned to the class instance
BringToTop	Show chart on top of other charts
EventObjectCreate	Sets a flag to send notifications of an event of new object creation on a chart
EventObjectDelete	Sets a flag to send notifications of an event of object deletion on a chart

Indicators	
IndicatorAdd	Adds an indicator with the specified handle into a specified chart subwindow
IndicatorDelete	Removes an indicator with a specified name from the specified chart subwindow
IndicatorsTotal	Returns the number of all indicators applied to the specified chart subwindow
IndicatorName	Returns the short name of the indicator on the specified chart subwindow
Navigation	
Navigate	Navigates the chart
Access to MQL5 API	
Symbol	Gets symbol of the chart
Period	Gets period of the chart
Redraw	Redraws chart, assigned to the class instance
GetInteger	The function returns the value of the corresponding object property
SetInteger	Sets new value for the property of the integer type
GetDouble	The function returns the value of the corresponding object property
SetDouble	Sets new value for the property of the double type
GetString	The function returns the value of the corresponding object property
SetString	Sets new value for the property of the string type
SetSymbolPeriod	Changes symbol and period of the chart, assigned to the class instance
ApplyTemplate	Applies specified template to the chart
ScreenShot	Creates screenshot of the specified chart and saves it to .gif file
WindowOnDropped	Gets chart subwindow number corresponding to the object (expert or script) drop point
PriceOnDropped	Gets price coordinate corresponding to the object (expert or script) drop point
TimeOnDropped	Gets time coordinate corresponding to the object (expert or script) drop point

XOnDropped	Gets X coordinate corresponding to the object (expert or script) drop point
YOnDropped	Gets Y coordinate corresponding to the object (expert or script) drop point
Input/Output	
virtual Save	Saves object parameters to file
virtual Load	Loads object parameters from file
virtual Type	Gets graphic object type identifier

ChartID

Returns identifier of the chart.

```
long  ChartID() const
```

Returned value

Chart identifier, assigned to the class instance. If there is no object assigned, it returns -1.

Mode (Get Method)

Gets the value of "Mode" property (bar, candle or line).

```
ENUM_CHART_MODE  Mode() const
```

Returned value

Value of "Mode" property of the object, assigned to the class instance. If there is no object assigned, it returns WRONG_VALUE.

Mode (Set Method)

Sets new value for "Mode" property (bar, candle or line).

```
bool  Mode(
   ENUM_CHART_MODE  mode       // new chart mode
   )
```

Parameters

mode

[in] Chart mode (candle, bar or line) of ENUM_CHART_MODE enumeration.

Returned value

true if successful, false if mode hasn't changed.

Foreground (Get Method)

Gets the value of "Foreground" property.

```
bool  Foreground() const
```

Returned value

Value of "Foreground" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

Foreground (Set Method)

Sets new value for "Foreground" property.

```
bool  Foreground(
   bool  foreground      // new flag value
   )
```

Parameters

foreground

[in] New value for "Foreground" property.

Returned value

true if successful, false if property hasn't changed.

Shift (Get Method)

Gets the value of "Shift" property.

```
bool  Shift() const
```

Returned value

Value of "Shift" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

Shift (Set Method)

Sets new value for "Shift" property.

```
bool  Shift(
   bool  shift      // new flag value
   )
```

Parameters

shift

[in] New value for "Shift" property.

Returned value

true if successful, false if property hasn't changed.

ShiftSize (Get Method)

Gets the value of "ShiftSize" property (in percents).

```
double  ShiftSize() const
```

Returned value

Value of "ShiftSize" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

ShiftSize (Set Method)

Sets new value for "Shift" property (in percents).

```
bool  ShiftSize(
   double  shift_size      // new property value
   )
```

Parameters

shift_size

[in] New value for "ShiftSize" property (in percents).

Returned value

true if successful, false if property hasn't changed.

AutoScroll (Get Method)

Gets the value of "AutoScroll" property.

```
bool  AutoScroll() const
```

Returned value

Value of "AutoScroll" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

AutoScroll (Set Method)

Sets new value for "AutoScroll" property.

```
bool  AutoScroll(
   bool  autoscroll      // new flag value
   )
```

Parameters

autoscroll

[in] New value for "Autoscroll" property.

Returned value

true if successful, false if property hasn't changed.

Scale (Get Method)

Gets the value of "Scale" property.

```
int  Scale() const
```

Returned value

Value of "Scale" property of the object, assigned to the class instance. If there is no object assigned, it returns 0.

Scale (Set Method)

Sets new value for "Scale" property.

```
bool  Scale(
   int  scale      // new value
   )
```

Parameters

scale

[in] New value for "Scale" property.

Returned value

true if successful, false if property hasn't changed.

ScaleFix (Get Method)

Gets the value of "ScaleFix" property (fixed chart scale or not).

```
bool  ScaleFix() const
```

Returned value

Value of "ScaleFix" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

ScaleFix (Set Method)

Sets new value for "ScaleFix" property.

```
bool  ScaleFix(
   bool  scale_fix      // new value
   )
```

Parameters

scale_fix

[in] New value for "ScaleFix" property.

Returned value

true if successful, false if property hasn't changed.

ScaleFix_11 (Get Method)

Gets the value of "ScaleFix_11" property (chart scale is 1:1, or not).

```
bool  ScaleFix_11() const
```

Returned value

Value of "ScaleFix_11" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

ScaleFix_11 (Set Method)

Sets new value for "ScaleFix_11" property.

```
bool  ScaleFix_11(
   string  scale_11      // new value
   )
```

Parameters

scale_11

[in] New value for "ScaleFix_11" property.

Returned value

true if successful, false if property hasn't changed.

FixedMax (Get Method)

Gets the value of "FixedMax" property (fixed maximal price).

```
double  FixedMax() const
```

Returned value

Value of "FixedMax" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

FixedMax (Set Method)

Sets new value for "FixedMax" property.

```
bool  FixedMax(
   double  max       // new fixed maximum
   )
```

Parameters

max

[in] New value for "FixedMax" property.

Returned value

true if successful, false if property hasn't changed.

FixedMin (Get Method)

Gets the value of "FixedMin" property (fixed minimal price).

```
double  FixedMin() const
```

Returned value

Value of "FixedMin" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

FixedMin (Set Method)

Sets new value for "FixedMin" property.

```
bool  FixedMax(
   double  min      // new fixed minimum
   )
```

Parameters

max

[in] New value for "FixedMin" property.

Returned value

true if successful, false if property hasn't changed.

PointsPerBar (Get Method)

Gets the value of "PointsPerBar" property (in points per bar).

```
double  PointsPerBar() const
```

Returned value

Value of "PointsPerBar" property of the object, assigned to the class instance. If there is no object assigned, it returns EMPTY_VALUE.

PointsPerBar (Set Method)

Sets new value for "PointsPerBar" property.

```
bool  PointsPerBar(
   double  ppb      // new scale (in points per bar)
   )
```

Parameters

ppb

[in] New value for scale (in points per bar).

Returned value

true if successful, false if scale hasn't changed.

ScalePPB (Get Method)

Gets the value of "ScalePPB" property (scale is "point per bar" or not).

```
bool  ScalePPB() const
```

Returned value

Value of "ScalePPB" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

ScalePPB (Set Method)

Sets new value for "ScalePPB" property.

```
bool  ScalePPB(
   bool  scale_ppb      // new flag value
   )
```

Parameters

scale_ppb

[in] New value for "ScalePPB" property.

Returned value

true if successful, false if property hasn't changed.

ShowOHLC (Get Method)

Gets the value of "ShowOHLC" property.

```
bool  ShowOHLC() const
```

Returned value

Value of "ShowOHLC" property of the object, assigned to the class instance. If there is no object assigned, it returns false.

ShowOHLC (Set Method)

Sets new value for "ShowOHLC" property.

```
bool  ShowOHLC(
   bool  show      // new value
   )
```

Parameters

show

[in] New value for "ShowOHLC" property.

Returned value

true if successful, false if property hasn't changed.

ShowLineBid (Get Method)

Gets the value of "ShowLineBid" property.

```
bool  ShowLineBid() const
```

Returned value

Value of "ShowLineBid" property of the chart, assigned to the class instance. If there is no chart assigned, it returns false.

ShowLineBid (Set Method)

Sets new value for "ShowLineBid" property.

```
bool  ShowLineBid(
   bool  show      // new value
   )
```

Parameters

show

[in] New value for "ShowLineBid" property.

Returned value

true if successful, false if property hasn't changed.

ShowLineAsk (Get Method)

Gets the value of "ShowLineAsk" property.

```
bool  ShowLineAsk() const
```

Returned value

Value of "ShowLineAsk" property of the chart, assigned to the class instance. If there is no chart assigned, it returns false.

ShowLineAsk (Set Method)

Sets new value for "ShowLineAsk" property.

```
bool  ShowLineAsk(
   bool  show      // new value
   )
```

Parameters

show

[in] New value for "ShowLineAsk" property.

Returned value

true if successful, false if property hasn't changed.

ShowLastLine (Get Method)

Gets the value of "ShowLastLine" property.

```
bool  ShowLastLine() const
```

Returned value

Value of "ShowLastLine" property of the chart, assigned to the class instance. If there is no chart assigned, it returns false.

ShowLastLine (Set Method)

Sets new value for "ShowLastLine" property.

```
bool  ShowLastLine(
   bool  show      // new flag value
   )
```

Parameters

show

[in] New value for "ShowLastLine" property.

Returned value

true if successful, false if property hasn't changed.

ShowPeriodSep (Get Method)

Gets the value of "ShowPeriodSep" property (show period separators).

```
bool  ShowPeriodSep() const
```

Returned value

Value of "ShowPeriodSep" property of the chart, assigned to the class instance. If there is no chart assigned, it returns false.

ShowPeriodSep (Set Method)

Sets new value for "ShowPeriodSep" property.

```
bool  ShowPeriodSep(
   bool  show      // new value
   )
```

Parameters

show

[in] New value for "ShowPeriodSep" property.

Returned value

true if successful, false if property hasn't changed.

ShowGrid (Get Method)

Gets the value of "ShowGrid" property.

```
bool  ShowGrid() const
```

Returned value

Value of "ShowGrid" property of the chart, assigned to the class instance. If there is no chart assigned, it returns false.

ShowGrid (Set Method)

Sets new value for "ShowGrid" property.

```
bool  ShowGrid(
   bool  show      // new value
   )
```

Parameters

show

[in] New value for "ShowGrid" property.

Returned value

true if successful, false if property hasn't changed.

ShowVolumes (Get Method)

Gets the value of "ShowVolumes" property.

```
bool  ShowVolumes() const
```

Returned value

Value of "ShowVolumes" property of the chart, assigned to the class instance. If there is no chart assigned, it returns false.

ShowVolumes (Set Method)

Sets new value for "ShowVolumes" property.

```
bool  ShowVolumes(
   bool  show      // new value
   )
```

Parameters

show

[in] New value for "ShowVolumes" property.

Returned value

true if successful, false if property hasn't changed.

ShowObjectDescr (Get Method)

Gets the value of "ShowObjectDescr" property (show description for graphic objects).

```
bool  ShowObjectDescr() const
```

Returned value

Value of "ShowObjectDescr" property of the chart, assigned to the class instance. If there is no chart assigned, it returns false.

ShowObjectDescr (Set Method)

Sets new value for "ShowObjectDescr" property.

```
bool  ShowObjectDescr(
   bool  show      // New value
   )
```

Parameters

show

[in] New value for "ShowObjectDescr" property.

Returned value

true if successful, false if property hasn't changed.

ShowDateScale

Sets new value for "ShowDateScale" property.

```
bool  ShowDateScale(
   bool  show      // New value
   )
```

Parameters

show

[in] New value for "ShowDateScale" property.

Returned value

true if successful, false if property hasn't changed.

ShowPriceScale

Sets new value for "ShowPriceScale" property.

```
bool  ShowPriceScale(
   bool  show      // New value
   )
```

Parameters

show

[in] New value for "ShowPriceScale" property.

Returned value

true if successful, false if property hasn't changed.

ColorBackground (Get Method)

Gets the value of "ColorBackground" property (background color of the chart).

```
color  ColorBackground() const
```

Returned value

Value of "ColorBackground" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorBackground (Set Method)

Sets new value for "ColorBackground" property.

```
bool  ColorBackground(
   color  new_color      // new background color
   )
```

Parameters

new_color

[in] New background color.

Returned value

true if successful, false if color hasn't changed.

ColorForeground (Get Method)

Gets the value of "ColorForeground" property (color of axes, scale and OHLC strings of the chart).

```
color  ColorForeground() const
```

Returned value

Value of "ColorForeground" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorForeground (Set Method)

Sets new value for "ColorForeground" property (for axes, scale, and OHLC string).

```
bool  ColorForeground(
   color  new_color      // New color
   )
```

Parameters

new_color

[in] New color for axes, scale and OHLC string.

Returned value

true if successful, false if color hasn't changed.

ColorGrid (Get Method)

Gets the value of "ColorGrid" property (color of the grid).

```
color  ColorGrid() const
```

Returned value

Value of "ColorGrid" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorGrid (Set Method)

Sets new value for "ColorGrid" property.

```
bool  ColorGrid(
   color  new_color      // new grid color
   )
```

Parameters

new_color

[in] New grid color.

Returned value

true if successful, false if color hasn't changed.

ColorBarUp (Get Method)

Gets the value of "ColorBarUp" property (color for bull bars, its shadow and candle body outlines).

```
color  ColorBarUp() const
```

Returned value

Value of "ColorBarUp" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorBarUp (Set Method)

Sets new value for "ColorBarUp" property.

```
bool  ColorBarUp(
   color  new_color      // new color for bull bars
   )
```

Parameters

new_color

[in] New color for bull bars.

Returned value

true if successful, false if color hasn't changed.

ColorBarDown (Get Method)

Gets the value of "ColorBarDown" property (color for bear bars, its shadow and candle body outlines).

```
color  ColorBarDown() const
```

Returned value

Value of "ColorBarDown" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorBarDown (Set Method)

Sets new value for "ColorBarDown" property.

```
bool  ColorBarDown(
   color  new_color      // new color for bear bars
   )
```

Parameters

new_color

[in] New color for bear bars.

Returned value

true if successful, false if color hasn't changed.

ColorCandleBull (Get Method)

Gets the value of "ColorCandleBull" property (body color of the bull candle).

```
color  ColorCandleBull() const
```

Returned value

Value of "ColorCandleBull" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorCandleBull (Set Method)

Sets new value for "ColorBarBull" property.

```
bool  ColorCandleBull(
   color  new_color      // new color for bull candle body
   )
```

Parameters

new_color

[in] New color of the bull candle body.

Returned value

true if successful, false if color hasn't changed.

ColorCandleBear (Get Method)

Gets the value of "ColorCandleBear" property (body color of the bear candle).

```
color  ColorCandleBear() const
```

Returned value

Value of "ColorCandleBear" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorCandleBear (Set Method)

Sets new value for "ColorBarBear" property.

```
bool  ColorCandleBear(
   color  new_color      // new color for bear candle body
   )
```

Parameters

new_color

[in] New color of the bear candle body.

Returned value

true if successful, false if color hasn't changed.

ColorChartLine (Get Method)

Gets the value of "ColorChartLine" property (color for line chart and Doji candles).

```
color  ColorChartLine() const
```

Returned value

Value of "ColorChartLine" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorChartLine (Set Method)

Sets new value for "ColorChartLine" property.

```
bool  ColorChartLine(
   color  new_color      // new color of the chart lines
   )
```

Parameters

new_color

[in] New color of the chart lines (Doji candles).

Returned value

true if successful, false if color hasn't changed.

ColorVolumes (Get Method)

Gets the value of "ColorVolumes" property (color for volumes and levels of opened positions).

```
color  ColorVolumes() const
```

Returned value

Value of "ColorVolumes" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorVolumes (Set Method)

Sets new value for "ColorVolumes" property.

```
bool  ColorVolumes(
   color  new_color      // new color of the volumes (open position levels)
   )
```

Parameters

new_color

[in] New color of the volumes (open position levels).

Returned value

true if successful, false if color hasn't changed.

ColorLineBid (Get Method)

Gets the value of "ColorLineBid" property (color of Bid line).

```
color  ColorLineBid() const
```

Returned value

Value of "ColorLineBid" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorLineBid (Set Method)

Sets new value for "ColorLineBid" property.

```
bool  ColorLineBid(
   color  new_color      // new color for Bid line
   )
```

Parameters

new_color

[in] New color for Bid line.

Returned value

true if successful, false if color hasn't changed.

ColorLineAsk (Get Method)

Gets the value of "ColorLineAsk" property (color of Ask line).

```
color  ColorLineAsk() const
```

Returned value

Value of "ColorLineAsk" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorLineAsk (Set Method)

Sets new value for "ColorLineAsk" property.

```
bool  ColorLineAsk(
   color  new_color      // new color for Ask line
   )
```

Parameters

new_color

[in] New color for Ask line.

Returned value

true if successful, false if color hasn't changed.

ColorLineLast (Get Method)

Gets the value of "ColorLineLast" property (color of the last deal price line).

```
color  ColorLineLast() const
```

Returned value

Value of "ColorLineLast" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorLineLast (Set Method)

Sets new value for "ColorLineLast" property.

```
bool  ColorLineLast(
   color  new_color      // new color of the last deal price line
   )
```

Parameters

new_color

[in] New color of the last deal price line.

Returned value

true if successful, false if color hasn't changed.

ColorStopLevels (Get Method)

Gets the value of "ColorStopLevels" property (color of the SL and TP levels).

```
color  ColorStopLevels() const
```

Returned value

Value of "ColorStopLevels" property of the chart, assigned to the class instance. If there is no chart assigned, it returns CLR_NONE.

ColorStopLevels (Set Method)

Sets new value for "ColorStopLevels" property.

```
bool  ColorStopLevels(
   color  new_color      // new color of the SL and TP price levels
   )
```

Parameters

new_color

[in] New color of the Stop Loss and Take Profit price levels.

Returned value

true if successful, false if color hasn't changed.

VisibleBars

Gets total number of visible chart bars.

```
int  VisibleBars() const
```

Returned value

Gets total number of visible bars of the chart, assigned to the class instance. If there is no chart assigned, it returns 0.

WindowsTotal

Gets total number of chart windows, including the chart indicator subwindows.

```
int  WindowsTotal() const
```

Returned value

Total number of windows, including the chart indicator subwindows, assigned to the class instance. If there is no chart assigned, it returns 0.

WindowIsVisible

Gets visibility flag of the specified chart subwindow.

```
bool  WindowIsVisible(
   int  num      // subwindow number
   ) const
```

Parameters

num

[in] Subwindow number (0 means base window).

Returned value

Returns visibility flag of the specified chart subwindow, assigned to the chart instance. If there is no chart assigned, it returns false.

WindowHandle

Gets window handle of the chart (HWND).

```
int  WindowHandle() const
```

Returned value

Window handle of the chart, assigned to the chart instance. If there is no chart assigned, it returns INVALID_HANDLE.

FirstVisibleBar

Gets the number of the first visible bar of the chart.

```
int  FirstVisibleBar() const
```

Returned value

Number of the first visible bar of the chart, assigned to the chart instance. If there is no chart assigned, it returns -1.

WidthInBars

Gets window width in bars.

```
int  WidthInBars() const
```

Returned value

Window width in chart bars, assigned to the chart instance. If there is no chart assigned, it returns 0.

WidthInPixels

Gets subwindow width in pixels.

```
int  WidthInPixels() const
```

Returned value

Subwindow width in chart pixels, assigned to the chart instance. If there is no chart assigned, it returns 0.

HeightInPixels

Gets subwindow height in pixels.

```
int  HeightInPixels(
   int  num      // subwindow number
   ) const
```

Parameters

num

[in] Subwindow number (0 means base window).

Returned value

Subwindow height in chart pixels, assigned to the chart instance. If there is no chart assigned, it returns 0.

PriceMin

Gets minimal price of the specified subwindow.

```
double  PriceMin(
   int  num      // subwindow number
   ) const
```

Parameters

num

[in] Subwindow number (0 means base window).

Returned value

Minimal price value of the chart, assigned to the class instance. If there is not chart assigned, it returns EMPTY_VALUE.

PriceMax

Gets maximal price of the specified subwindow.

```
double  PriceMax(
   int  num      // subwindow number
   ) const
```

Parameters

num

[in] Subwindow number (0 means base window).

Returned value

Maximal price value of the chart, assigned to the class instance. If there is not chart assigned, it returns EMPTY_VALUE.

Attach

Assigns the current chart to the class instance.

```
void  Attach()
```

Attach

Assigns the specified chart to the class instance.

```
void  Attach(
   long  chart      // Chart identifier
   )
```

Parameters

chart

[in] Identifier of the chart to assign.

FirstChart

Assigns the first chart of the client terminal to the class instance.

```
void  FirstChart()
```

NextChart

Assigns the next chart of the client terminal to the class instance.

```
void  NextChart()
```

Open

Opens chart with specified parameters and assign it to the class instance.

```
long  Open(
   const string      symbol_name,      // Symbol name
   ENUM_TIMEFRAMES   timeframe         // Period
   )
```

Parameters

symbol_name

[in] Symbol name. NULL means the symbol of the current chart (to which expert attached).

timeframe

[in] Chart timeframe (ENUM_TIMEFRAMES enumeration). 0 means the current timeframe.

Returned value

Chart identifier.

Detach

Detaches chart from the class instance.

```
void  Detach()
```

Close

Closes chart, assigned to the class instance.

```
void  Close()
```

BringToTop

Show chart on top of other charts.

```
bool  BringToTop() const
```

Returned value

true if successful, false if error.

EventObjectCreate

Sets a flag to send notifications of an event of new object creation to all MQL5-programs on a chart.

```
bool  EventObjectCreate(
   bool  flag      // flag
   )
```

Parameters

flag

[in] New flag value.

Returned value

true - if successful, false - if flag hasn't been changed.

EventObjectDelete

Sets a flag to send notifications of an event of object deletion to all MQL5-programs on a chart.

```
bool  EventObjectDelete(
   bool  flag      // flag
   )
```

Parameters

flag

[in] New flag value.

Returned value

true - if successful, false - if flag hasn't been changed.

IndicatorAdd

Adds an indicator with the specified handle into a specified chart window.

```
bool  IndicatorAdd(
   int   sub_win          // number of the sub-window
   int   handle           // handle of the indicator
   );
```

Parameters

sub_win

[in] The number of the chart sub-window. 0 means the main chart window. if the number of a not-existing window is specified, a new window will be created.

handle

[in] The handle of the indicator.

Return Value

The function returns true in case of success, otherwise it returns false. In order to obtain information about the error, call the GetLastError() function.

See Also

IndicatorDelete(), IndicatorsTotal(), IndicatorName().

IndicatorDelete

Removes an indicator with a specified name from the specified chart window.

```
bool  IndicatorDelete(
   int            sub_win       // number of the subwindow
   const string   name          // short name of the indicator
   );
```

Parameters

sub_win

[in] Number of the chart subwindow. 0 denotes the main chart subwindow.

const name

[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 IndicatorName() function.

Return Value

Returns true in case of successful deletion of the indicator. Otherwise it returns 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.

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions iCustom() and IndicatorCreate(). 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.

Deletion of an indicator from a chart doesn't mean that its calculation part will be deleted from the terminal memory. To release the indicator handle use the IndicatorRelease() function.

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 IndicatorDelete() function is identified by the short name.

See also

IndicatorAdd(), IndicatorsTotal(), IndicatorName(), iCustom(), IndicatorCreate(), IndicatorSetString().

IndicatorsTotal

Returns the number of all indicators applied to the specified chart window.

```
int  IndicatorsTotal(
   int   sub_win     // number of the subwindow
   );
```

Parameters

sub_win

[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 GetInteger() function.

See also

IndicatorAdd(), IndicatorDelete(), IndicatorsTotal(), iCustom(), IndicatorCreate(), IndicatorSetString().

IndicatorName

Returns the short name of the indicator by the index in the indicators list on the specified chart window.

```
string  IndicatorName(
   int    sub_win      // number of the subwindow
   int    index        // index of the indicator in the list of indicators added to the
   );
```

Parameters

sub_win

[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 IndicatorsTotal() 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

Do not confuse the indicator short name and the file name that is specified when creating an indicator using functions iCustom() and IndicatorCreate(). 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 IndicatorDelete() function is identified by the short name.

See also

IndicatorAdd(), IndicatorDelete, IndicatorsTotal, iCustom(), IndicatorCreate(), IndicatorSetString().

Navigate

Navigates the chart.

```
bool  Navigate(
   ENUM_CHART_POSITION  position,      // Position
   int                  shift=0       // Shift
   )
```

Parameters

position

[in] Value of ENUM_CHART_POSITION enumeration.

shift=0

[in] Number of bars to shift.

Returned value

true if successful, false if chart hasn't navigated.

Symbol

Gets symbol of the chart.

```
string  Symbol() const
```

Returned value

Symbol of the chart, assigned to the class instance. If there is no chart assigned, it returns 0.

Period

Gets period of the chart.

```
ENUM_TIMEFRAMES  Period() const
```

Returned value

Period of the chart, assigned to the class instance. If there is no chart assigned, it returns 0.

Redraw

Redraws chart, assigned to the class instance.

```
void  Redraw()
```

GetInteger

The function returns the value of the corresponding object property. The object property must be of the integer type. There are 2 variants of the function.

1. Immediately returns the property value.

```
long  GetInteger(
   ENUM_CHART_PROPERTY_INTEGER  prop_id,          // property identifier
   int                          sub_window=0      // subwindow number
   ) const
```

2. If successful, puts the value of property to the specified variable of integer type, passed by reference as last parameter.

```
bool  GetInteger(
   ENUM_CHART_PROPERTY_INTEGER  prop_id,        // property identifier
   int                          sub_window,     // subwindow number
   long&                        value           // here we get the property value
   ) const
```

Parameters

prop_id

[in] Property identifier (ENUM_CHART_PROPERTY_INTEGER enumeration).

sub_window

[in] Chart subwindow number.

value

[in] Variable of the integer type that received the value of the requested property.

Return Value

Value of property of the chart, assigned to the class instance. If there isn't any chart assigned, it returns -1.

For the second variant the function returns true, if this property is maintained and the value has been placed into the value variable, otherwise returns false. To read more about the error call GetLastError().

SetInteger

Sets new value for the property of the integer type.

```
bool  SetInteger(
   ENUM_CHART_PROPERTY_INTEGER  prop_id,      // property identifier
   long                         value         // new value
   )
```

Parameters

prop_id

[in] Property identifier (ENUM_CHART_PROPERTY_INTEGER enumeration).

value

[in] New value of the property.

Returned value

true if successful, false if property of the integer type hasn't changed.

GetDouble

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  GetDouble(
   ENUM_CHART_PROPERTY_DOUBLE  prop_id,          // property identifier
   int                         sub_window=0     // subwindow number
   ) const
```

2. If successful, puts the value of property to the specified variable of double type, passed by reference as last parameter.

```
bool  GetDouble(
   ENUM_CHART_PROPERTY_DOUBLE  prop_id,        // property identifier
   int                         sub_window,     // subwindow number
   double&                     value           // here we get the property value
   ) const
```

Parameters

prop_id

[in] Property identifier (ENUM_CHART_PROPERTY_DOUBLE enumeration).

sub_window

[in] Chart subwindow number.

value

[in] Variable of the double type that received the value of the requested property.

Return Value

Value of property of the chart, assigned to the class instance. If there isn't any chart assigned, it returns EMPTY_VALUE.

For the second variant the function returns true, if this property is maintained and the value has been placed into the value variable, otherwise returns false. To read more about the error call GetLastError().

SetDouble

Sets new value for the property of the double type.

```
bool  SetDouble(
   ENUM_CHART_PROPERTY_DOUBLE  prop_id,      // property identifier
   double                      value         // new value
   )
```

Parameters

prop_id

[in] Property idenitifier (ENUM_CHART_PROPERTY_DOUBLE enumeration).

value

[in] New value for the property.

Returned value

true if successful, false if property of the double type hasn't changed.

GetString

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  GetString(
   ENUM_CHART_PROPERTY_STRING  prop_id      // property identifier
   ) const
```

2. If successful, puts the value of property to the specified variable of string type, passed by reference as last parameter.

```
bool  GetString(
   ENUM_CHART_PROPERTY_STRING  prop_id,     // property identifier
   string&                     value        // here we get the property value
   ) const
```

Parameters

prop_id

[in] Property identifier (ENUM_CHART_PROPERTY_STRING enumeration).

sub_window

[in] Chart subwindow number.

value

[in] Variable of the string type that received the value of the requested property.

Return Value

Value of property of the chart, assigned to the class instance. If there isn't any chart assigned, it returns "".

For the second variant the function returns true, if this property is maintained and the value has been placed into the value variable, otherwise returns false. To read more about the error call GetLastError().

SetString

Sets new value for the property of the string type.

```
bool  SetString(
   ENUM_CHART_PROPERTY_STRING  prop_id,      // property identifier
   string                      value         // new property value
   )
```

Parameters

prop_id

[in] Property idenitifier (ENUM_CHART_PROPERTY_STRING enumeration).

value

[in] New value for the property.

Returned value

true if successful, false if property of the string type hasn't changed.

SetSymbolPeriod

Changes symbol and period of the chart, assigned to the class instance.

```
bool  SetSymbolPeriod(
   const string      symbol_name,      // Symbol
   ENUM_TIMEFRAMES   timeframe         // Period
   )
```

Parameters

symbol_name

[in] New symbol name. NULL means the symbol of the current chart (to which expert attached).

timeframe

[in] New chart timeframe (ENUM_TIMEFRAMES enumeration). 0 means the current timeframe.

Returned value

true if successful, false if property hasn't changed.

ApplyTemplate

Applies specified template to the chart.

```
bool  ApplyTemplate(
   const string  filename      // template file name
   )
```

Parameters

filename

[in] File name of the template.

Returned value

true if successful, false if template hasn't applied.

ScreenShot

Creates screenshot of the specified chart and saves it to .gif file.

```
bool  ScreenShot(
   string           filename,                  // File name
   int              width,                     // Width
   int              height,                    // Height
   ENUM_ALIGN_MODE  align_mode=ALIGN_RIGHT     // Align type
   ) const
```

Parameters

filename

[in] File name for screenshot.

width

[in] Screenshot width in pixels.

height

[in] Screenshot height in pixels.

align_mode=ALIGN_RIGHT

[in] Align mode, if screenshot is narrow.

Returned value

true if successful, false if error.

WindowOnDropped

Gets chart subwindow number corresponding to the object (expert or script) drop point.

```
int  WindowOnDropped() const
```

Returned value

Chart subwindow number of the object drop point. 0 means main chart window.

PriceOnDropped

Gets price coordinate corresponding to the object (expert or script) drop point.

```
double  PriceOnDropped() const
```

Returned value

Price coordinate of the object drop point.

TimeOnDropped

Gets time coordinate corresponding to the object (expert or script) drop point.

```
datetime  TimeOnDropped() const
```

Returned value

Time coordinate of the object drop point.

XOnDropped

Gets X coordinate corresponding to the object (expert or script) drop point.

```
int  XOnDropped() const
```

Returned value

X coordinate of the object drop point.

YOnDropped

Gets Y coordinate corresponding to the object (expert or script) drop point.

```
int  YOnDropped() const
```

Returned value

Y coordinate of the object drop point.

Save

Saves object parameters to file.

```
virtual bool  Save(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen(...) function.

Returned value

true if successful, false if error.

Load

Loads object parameters from file.

```
virtual bool  Load(
   int  file_handle      // File handle
   )
```

Parameters

file_handle

[in] handle of the binary file already opened by FileOpen(...) function.

Returned value

true if successful, false if error.

Type

Returns graphic object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (0x1111 for CChart).

File Operations

This section contains the technical details of the file operations classes and descriptions of the corresponding components of the standard MQL5 library.

The file operations classes use will save time in developing applications which uses file input/output operations.

The MQL5 Standard Library is placed in the working directory of the terminal in the Include\Files folder.

Class	Description
CFile	Base file operations class
CFileBin	Binary file operations class
CFileTxt	Text file operations class

CFile

CFile is a base class for CFileBin and CFileTxt classes.

Description

Class CFile provides the simplified access for all of its descendants to MQL5 API file and folder functions.

Declaration

```
class CFile: public CObject
```

Title

```
#include <Files\File.mqh>
```

Class Methods

Attributes	
Handle	Gets file handle
Filename	Gets file name
Flags	Gets file flags
SetUnicode	Sets/Clears the FILE_UNICODE flag
SetCommon	Sets/Clears the FILE_COMMON flag
General methods for files	
Open	Opens file
Close	Closes file
Delete	Deletes file
IsExist	Checks file for existence
Copy	Copies file
Move	Renames/moves file
Size	Gets file size
Tell	Gets current file position
Seek	Sets current file position
Flush	Flushes data on disk
IsEnding	Checks file for end
IsLineEnding	Checks line for end
General methods for folders	

FolderCreate	Creates folder
FolderDelete	Deletes folder
FolderClean	Clears folder
Search methods	
FileFindFirst	Begin file search
FileFindNext	Continue file search
FileFindClose	Close search handle

Derived classes:

- CFileBin
- CFileTxt

Handle

Gets file handle of the opened file.

```
int  Handle()
```

Returned value

Handle of the opened file, assigned to the class instance. If there is no file assigned, it returns -1.

FileName

Gets file name of the opened file.

```
string  FileName()
```

Returned value

File name of the opened file, assigned to the class instance. If there is no file assigned, it returns "".

Flags

Gets flags of the opened file.

```
int  Flags()
```

Returned value

Flags of the opened file, assigned to the class instance.

SetUnicode

Sets/Clears the FILE_UNICODE flag.

```
void  SetUnicode(
   bool  unicode      // New flag value
   )
```

Parameters

unicode

[in] New value for FILE_UNICODE flag.

Note

The result of string operations is dependent on the FILE_UNICODE flag. If it's false, the ANSI codes are used (one byte symbols). If it set, the UNICODE codes are used (two byte symbols). If the file has already opened, the flag cannot be changed.

SetCommon

Sets/Clears the FILE_COMMON flag.

```
void  SetCommon(
   bool  common      // New flag value
   )
```

Parameters

common

[in] New value for FILE_COMMON flag.

Note

The FILE_UNICODE flag determines the current work folder. If it's false, the local terminal folder used as the current work folder. If it's true, the general folder used as the current work folder. If the file has already opened, the flag cannot be changed.

Open

Open the specified file and if it successful, assigns it to the class instance.

```
int  Open(
   const string  file_name,        // File name
   int           flags,            // Flags
   short         delimiter=9       // Separator
   )
```

Parameters

file_name

[in] File name to open.

flags

[in] File open flags.

delimiter=9

[in] CSV file separator.

Returned value

Handle of the opened file.

Note

The work folder is dependent on the FILE_COMMON flag, defined by SetCommon() method.

Close

Closes file, assigned to the class instance.

```
void  Close()
```

Delete

Deletes the file, assigned to the file instance.

```
void  Delete()
```

Delete

Deletes the specified file.

```
void  Delete(
   const string  file_name      // File name
   )
```

Parameters

file_name

[in] File name of the file to delete.

Note

The work folder is dependent on FILE_COMMON flag, defined by SetCommon() method.

IsExist

Checks file for existence

```
bool  IsExist(
   const string  file_name      // File name
   )
```

Parameters

file_name

[in] Name of the file to check.

Returned value

true, if file exists.

Copy

Copies a file.

```
bool  Copy(
   const string  src_name,      // Source file name
   int           src_flag,      // Flag
   const string  dst_name,      // Destination file name
   int           dst_flags      // Flags
   )
```

Parameters

src_name

[in] File name of the file to copy.

src_flag

[in] Flags of the file to copy (only FILE_COMMON is used).

dst_name

[in] File name of the destination file.

dst_flags

[in] Flags of the destination file (only FILE_REWRITE and FILE_COMMON are used).

Returned value

true if successful, false if it hasn't been copied.

Move

Renames/moves file.

```
bool  Move(
   const string  src_name,      // Source file name
   int           src_flag,      // Flag
   const string  dst_name,      // Destination file name
   int           dst_flags      // Flags
   )
```

Parameters

src_name

[in] File name of the file to move.

src_flag

[in] Flags of the file to copy (only FILE_COMMON is used).

dst_name

[in] File name of the destination file.

dst_flags

[in] Flags of the destination file (only FILE_REWRITE and FILE_COMMON are used).

Returned value

true if successful, false if it hasn't been moved.

Size

Gets file size in bytes.

```
ulong  Size()
```

Returned value

File size in bytes. If there isn't any file assigned, it returns ULONG_MAX.

Tell

Gets the current file position.

```
ulong  Tell()
```

Returned value

The current file position. If there isn't any file assigned, it returns ULONG_MAX.

Seek

Sets current file position.

```
void  Seek(
   long                offset,     // Offset
   ENUM_FILE_POSITION  origin      // Origin
   )
```

Parameters

offset

[in] File offset in bytes (can be negative).

origin

[in] Origin of the offset.

Returned value

true if successful, false if file position hasn't been changed.

Flush

Flushes all of the file input/output buffer data on disk.

```
void  Flush()
```

IsEnding

Checks file for end. It's used during the file read operations.

```
bool  IsEnding()
```

Returned value

true if end of file has been achieved after read or seek operation.

IsLineEnding

Checks file for end of line. It's used during the file read operations.

```
bool  IsLineEnding()
```

Returned value

true if end of line has been achieved after the txt or csv file read operation (CR-LF chars).

FolderCreate

Creates new folder.

```
bool  FolderCreate(
   const string  folder_name      // Folder name
   )
```

Parameters

folder_name

[in] Name of the folder to create. It contains path to the folder relative to the folder defined by FILE_COMMON flag.

Returned value

true if successful, and false if the folder hasn't been created.

Note

The work folder is dependent on FILE_COMMON flag, defined by SetCommon() method.

FolderDelete

Deletes specified folder.

```
bool  FolderDelete(
   const string  folder_name      // Folder name
   )
```

Parameters

folder_name

[in] Name of the folder to delete. It contains path to the folder relative to the folder defined by FILE_COMMON flag.

Returned value

true if successful, and false if the folder hasn't been deleted.

Note

The work folder is dependent on FILE_COMMON flag, defined by SetCommon() method.

FolderClean

Cleans specified folder.

```
bool  FolderClean(
   const string  folder_name      // Folder name
   )
```

Parameters

folder_name

[in] Name of the folder to delete. It contains path to the folder relative to the folder defined by FILE_COMMON flag.

Returned value

true if successful, and false if the folder hasn't been cleaned.

Note

The work folder is dependent on FILE_COMMON flag, defined by SetCommon() method.

FileFindFirst

It began file search using the filter specified.

```
int  FileFindFirst(
   const string  filter,        // Search Filter
   string&       file_name      // Reference to string
   )
```

Parameters

filter

[in] Search filter.

file_name

[out] The reference to string for the first file found.

Returned value

If successful, it returns handle, that can be used for further file search using FileFindNext, or INVALID_HANDLE if there isn't any file corresponding to the filter specified.

Note

The work folder is dependent on FILE_COMMON flag, defined by SetCommon() method.

FileFindNext

It continue search, started by function FileFindFirst().

```
bool  FileFindNext(
   int      search_handle,     // Search handle
   string&  file_name          // Reference to string for the next file found
   )
```

Parameters

search_handle

[in] Search handle, returned by FileFindFirst() method.

file_name

[in] The reference to string for the name of the file found if successful.

Returned value

true if successful, false if there isn't any file, corresponding to the filter specified.

FileFindClose

Closes search handle.

```
void  FileFindClose(
   int  search_handle      // Search handle
   )
```

Parameters

search_handle

[in] Search handle, returned by FileFindFirst() method.

CFileBin

CFileBin is a class for simplified access to binary files.

Description

Class CFileBin provides an access to binary files.

Declaration

```
class CFileBin: public CFile
```

Title

```
#include <Files\FileBin.mqh>
```

Class Methods

Open methods	
Open	Opens a binary file
Write methods	
WriteChar	Writes char or uchar type variable
WriteShort	Writes short or ushort type variable
WriteInteger	Writes int or uint type variable
WriteLong	Writes long or ulong type variable
WriteFloat	Writes float type variable
WriteDouble	Writes double type variable
WriteString	Writes string type variable
WriteCharArray	Writes an array of char or uchar type variables
WriteShortArray	Writes an array of short or ushort type variables
WriteIntegerArray	Writes an array of int or uint type variables
WriteLongArray	Writes an array of long or ulong type variables
WriteFloatArray	Writes an array of float variables
WriteDoubleArray	Writes an array of double type variables
WriteObject	Writes data of the CObject class inheritor instance
Read methods	
ReadChar	Reads char or uchar type variable
ReadShort	Reads short or ushort type variable

ReadInteger	Reads int or uint type variable
ReadLong	Reads long or ulong type variable
ReadFloat	Reads float type variable
ReadDouble	Reads double type variable
ReadString	Reads string type variable
ReadCharArray	Reads an array of char or uchar type variables
ReadShortArray	Reads an array of short or ushort type variables
ReadIntegerArray	Reads an array of int or uint type variables
ReadLongArray	Reads an array of long or ulong type variables
ReadFloatArray	Reads an array of float type variables
ReadDoubleArray	Reads an array of double type variables
ReadObject	Reads data of the CObject class inheritor instance

Open

Open the specified binary file and if it successful, assigns it to the class instance.

```
int  Open(
   const string  file_name,      // File name
   int           flags           // Flags
   )
```

Parameters

file_name

[in] File name of the file to open.

flags

[in] File open flags (there is a forced set of the FILE_BIN flag).

Returned value

Handle of the opened file.

WriteChar

Writes char or uchar type variable to file.

```
uint  WriteChar(
   char  value      // Value
   )
```

Parameters

value

[in] Variable to write.

Returned value

Number of bytes written.

WriteShort

Writes short or ushort type variable to file.

```
uint  WriteShort(
   short  value      // Value
   )
```

Parameters

value

[in] Variable to write.

Returned value

Number of bytes written.

WriteInteger

Writes int or uint type variable to file.

```
uint  WriteInteger(
   int  value      // Value
   )
```

Parameters

value

[in] Variable to write.

Returned value

Number of bytes written.

WriteLong

Writes long or ulong type variable to file.

```
uint  WriteLong(
   long  value      // Value
   )
```

Parameters

value

[in] Variable to write.

Returned value

Number of bytes written.

WriteFloat

Writes float type variable to file.

```
uint  WriteFloat(
   float  value      // Value
   )
```

Parameters

value

[in] Variable to write.

Returned value

Number of bytes written.

WriteDouble

Writes double type variable to file.

```
uint  WriteDouble(
   double  value      // Value
   )
```

Parameters

value

[in] Variable to write.

Returned value

Number of bytes written.

WriteString

Writes string type variable to file.

```
uint  WriteString(
   const string  value      // Value
   )
```

Parameters

value

[in] String to write.

Returned value

Number of bytes written.

WriteString

Writes string type variable to file.

```
uint  WriteString(
   const string  value,     // Value
   int           size       // Size
   )
```

Parameters

value

[in] String to write.

size

[in] Number of bytes to write.

Returned value

Number of bytes written.

WriteCharArray

Writes an array of char or uchar type variables to file.

```
uint  WriteCharArray(
   char&  array[],            // Array reference
   int    start_item=0,       // Start element
   int    items_count=-1      // Number of elements
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - for whole array).

Returned value

Number of bytes written.

WriteShortArray

Writes an array of short or ushort type variables to file.

```
uint  WriteShortArray(
   short&  array[],            // Array to write
   int     start_item=0,       // Start element
   int     items_count=-1      // Number of elements to write
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - for whole array).

Returned value

Number of bytes written.

WriteIntegerArray

Writes an array of int or uint type variables to file.

```
uint  WriteIntegerArray(
   int&  array[],            // Array to write
   int   start_item=0,       // Start element
   int   items_count=-1      // Number of elements to write
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - for whole array).

Returned value

Number of bytes written.

WriteLongArray

Writes an array of long or ulong type variables to file.

```
uint  WriteLongArray(
   long&  array[],            // Array to write
   int    start_item=0,       // Start element
   int    items_count=-1      // Number of elements to write
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - for whole array).

Returned value

Number of bytes written.

WriteFloatArray

Writes an array of float type variables to file.

```
uint  WriteFloatArray(
   float&  array[],            // Array to write
   int     start_item=0,       // Start element
   int     items_count=-1      // Number of elements to write
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - for whole array).

Returned value

Number of bytes written.

WriteDoubleArray

Writes an array of double type variables to file.

```
uint  WriteDoubleArray(
   double&  array[],            // Array to write
   int      start_item=0,       // Start element
   int      items_count=-1      // Number of elements to write
   )
```

Parameters

array[]

[in] Array to write.

start_item=0

[in] Start element to write from.

items_count=-1

[in] Number of elements to write (-1 - for whole array).

Returned value

Number of bytes written.

WriteObject

Writes data of the CObject class inheritor instance to file.

```
bool  WriteObject(
   CObject*  object      // Reference to the object
   )
```

Parameters

object

[in] Reference to the CObject class inheritor instance to write.

Returned value

true if successful, false if data hasn't been written.

ReadChar

Reads char or uchar type variable from file.

```
bool  ReadChar(
   char&  value      // Target variable
   )
```

Parameters

value

[in] Target variable of type char.

Returned value

true if successful, false if data hasn't been read.

ReadShort

Reads short or ushort type variable from file.

```
bool  ReadShort(
   short&  value
   )
```

Parameters

value

[in] Target variable of type short or ushort.

Returned value

true if successful, false if data hasn't been read.

ReadInteger

Reads int or uint type variable from file.

```
bool  ReadInteger(
   int&  value      // Target variable
   )
```

Parameters

value

[in] Target variable of type int or uint.

Returned value

true if successful, false if data hasn't been read.

ReadLong

Reads long or ulong type variable from file.

```
bool  ReadLong(
   long&  value
   )
```

Parameters

value

[in] Target variable of type long or ulong.

Returned value

true if successful, false if data hasn't been read.

ReadFloat

Reads float type variable from file.

```
bool  ReadFloat(
   float&  value      // Target variable
   )
```

Parameters

value

[in] Target variable of type float.

Returned value

true if successful, false if data hasn't been read.

ReadDouble

Reads double type variable from file.

```
bool  ReadDouble(
   double&  value
   )
```

Parameters

value

[in] Target variable of type double.

Returned value

true if successful, false if data hasn't been read.

ReadString

Reads string type variable from file.

```
bool  ReadString(
   string&  value      // Target string
   )
```

Parameters

value

[in] Target variable of type string.

Returned value

true if successful, false if data hasn't been read.

ReadString

Reads string type variable from file.

```
bool  ReadString(
   string&  value
   )
```

Parameters

value

[in] Target variable of type string.

Returned value

true if successful, false if data hasn't been read.

ReadCharArray

Reads an array of char or uchar type variables from file.

```
bool  ReadCharArray(
   char&  array[],            // Target array
   int    start_item=0,       // Start element
   int    items_count=-1      // Number of elements to read
   )
```

Parameters

array[]

[in] Reference to the target array of type char or uchar.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Returned value

true if successful, false if data hasn't been read.

ReadShortArray

Reads an array of short or ushort type variables from file.

```
bool  ReadShortArray(
   short&  array[],            // Target array
   int     start_item=0,       // Start element
   int     items_count=-1      // Number of elements to read
   )
```

Parameters

array[]

[in] Reference to the target array of type short or ushort.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Returned value

true if successful, false if data hasn't been read.

ReadIntegerArray

Reads an array of int or uint type variables from file.

```
bool  ReadIntegerArray(
   int&  array[],            // Target array
   int   start_item=0,       // Start element
   int   items_count=-1      // Number of elements to read
   )
```

Parameters

array[]

[in] Reference to the target array of type int or uint.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Returned value

true if successful, false if data hasn't been read.

ReadLongArray

Reads an array of long or ulong type variables from file.

```
bool  ReadLongArray(
   long&  array[],            // Target array
   int    start_item=0,       // Start element
   int    items_count=-1      // Number of elements to read
   )
```

Parameters

array[]

[in] Reference to the target array of type long or ulong.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Returned value

true if successful, false if data hasn't been read.

ReadFloatArray

Reads an array of float type variables from file.

```
bool  ReadFloatArray(
   float&  array[],            // Target array
   int     start_item=0,       // Start element
   int     items_count=-1      // Number of elements to read
   )
```

Parameters

array[]

[in] Reference to the target array of type float.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Returned value

true if successful, false if data hasn't been read.

ReadDoubleArray

Reads an array of double type variables from file.

```
bool  ReadDoubleArray(
   double&  array[],            // Target array
   int      start_item=0,       // Start element
   int      items_count=-1      // Number of elements to read
   )
```

Parameters

array[]

[in] Reference to the target array of type double.

start_item=0

[in] Start element to read from.

items_count=-1

[in] Number of elements to read (-1 - read to the end of file).

Returned value

true if successful, false if data hasn't been read.

ReadObject

Reads data of the CObject class inheritor instance from file.

```
bool  ReadObject(
   CObject*  object      // Reference to the object
   )
```

Parameters

object

[in] Reference to the target CObject class inheritor instance for read to.

Returned value

true if successful, false if data hasn't been read.

CFileTxt

CFileTxt is a class for simplified access to text files.

Description

Class CFileTxt provides an access to text files.

Declaration

```
class CFileTxt: public CFile
```

Title

```
#include <Files\FileTxt.mqh>
```

Class Methods

Open methods	
Open	Open a text file
Write methods	
WriteString	Writes string type variable to file
Read methods	
ReadString	Reads string type variable from file

Open

Open the specified text file and if it successful, assigns it to the class instance.

```
int  Open(
   const string  file_name,      // file name
   int           flags           // flags
   )
```

Parameters

file_name

[in] File name to open.

flags

[in] File open flags (there is a forced set of the FILE_TXT flag).

Returned value

Opened file handle.

WriteString

Writes string type variable to file.

```
uint  WriteString(
   const string  value      // String to write
   )
```

Parameters

value

[in] String to write.

Returned value

Number of bytes written.

ReadString

Reads string type variable from file.

```
string  ReadString()
```

Returned value

String which has been read.

String operations

This section contains the technical details of the string operations classes and descriptions of the corresponding components of the standard MQL5 library.

The use of string operations classes will save time in developing applications which uses text processing operations.

The MQL5 Standard Library is placed in the working directory of the terminal in the Include\Strings folder.

Class	Description
CString	Class for string operations

CString

CString is a class for simplified access to the variables of string type.

Description

Class CFile provides the simplified access for all of its descendants to MQL5 API string functions.

Declaration

```
class CString: public CObject
```

Title

```
#include <Strings\String.mqh>
```

Class Methods

Data access methods	
Str	Gets a string
Len	Gets length of a string
Copy	Copies a string
Fill methods	
Fill	Fills a string with specified char
Assign	Assigns a string
Append	Appends a string
Insert	Inserts a string
Compare methods	
Compare	Compares a string
CompareNoCase	Compares a strings case insensitive
Substring methods	
Left	Gets a specified number of characters from the left side of a string
Right	Gets a specified number of characters from the right side of a string
Mid	Gets a specified number of characters from a string
Trim/delete methods	
Trim	Removes all leading and trailing occurrences of a set of specified characters from a string

TrimLeft	Removes all leading occurrences of a set of specified characters from a string
TrimRight	Removes all trailing occurrences of a set of specified characters from a string
Clear	Clears a string
Convert methods	
ToUpper	Converts a string to uppercase.
ToLower	Converts a string to lowercase.
Reverse	Reverses a string
Search methods	
Find	Searches for the first match of a substring
FindRev	Searches for the last match of a substring
Remove	Deletes a substring from a string
Replace	Replaces a substring

Str

Gets a string.

```
string  Str() const;
```

Returned value

Copy of a string.

Len

Gets length of a string.

```
uint  Len() const;
```

Returned value

Length of a string.

Copy

Copies a string by reference.

```
void  Copy(
   string&  copy      // Reference
   ) const;
```

Parameters

copy

[in] Reference to a string to copy.

Copy

Copies a string to the CString class instance.

```
void  Copy(
   CString*  copy      // Object descriptor
   ) const;
```

Parameters

copy

[in] CString class object descriptor.

Fill

Fills a string with specified char.

```
bool  Fill(
   short  character      // Character
   )
```

Parameters

character

[in] Character for filling.

Returned value

true if successful, false if a string hasn't been filled.

Assign

Assigns a string.

```
void  Assign(
   const string  str      // String to assign
   )
```

Parameters

str

[in] String to assign.

Assign

Assigns a string to the CString class instance.

```
void  Assign(
   CString*  str      // Object descriptor
   )
```

Parameters

str

[in] CString class object descriptor to assign.

Append

Appends a string.

```
void  Append(
   const string  str      // String to append
   )
```

Parameters

str

[in] String to append.

Append

Appends a string to the CString class instance.

```
void  Append(
   CString*  string      // Object descriptor
   )
```

Parameters

string

[in] CString class object descriptor to append.

Insert

Inserts a string to the specified position.

```
uint  Insert(
   uint          pos,     // Position
   const string  str      // String to insert
   )
```

Parameters

pos

[in] Insert position.

str

[in] String to insert.

Returned value

Resulted string length.

Insert

Inserts a string to the specified position to the CString class instance.

```
uint  Insert(
   uint      pos,     // Position
   CString*  str      // Object descriptor
   )
```

Parameters

pos

[in] Insert position.

str

[in] CString class object descriptor to insert.

Returned value

Resulted string length.

Compare

Compares a string.

```
int  Compare(
   const string  str      // String to compare
   ) const;
```

Parameters

str

[in] String to compare.

Returned value

I returns 0 if a strings are equal, -1 if a string of the class is lower than a string to compare, -1 if the class string greater than a string to compare.

Compare

Compares a string with a string of the CString class instance.

```
int  Compare(
   CString*  str      // Object descriptor
   ) const;
```

Parameters

str

[in] CString class object descriptor to compare.

Returned value

I returns 0 if a strings are equal, -1 if a string of the class is lower than a string to compare, -1 if the class string greater than a string to compare.

CompareNoCase

Compares a strings case insensitive.

```
int  CompareNoCase(
   const string  str      // String to compare
   ) const;
```

Parameters

str

[in] String to compare.

Returned value

I returns 0 if a strings are equal, -1 if a string of the class is lower than a string to compare, -1 if the class string greater than a string to compare.

CompareNoCase

Compares a string (case insensitive) with a string of the CString class instance.

```
int  CompareNoCase(
   CString*  str      // Object descriptor
   ) const;
```

Parameters

str

[in] CString class object descriptor to compare.

Returned value

I returns 0 if a strings are equal, -1 if a string of the class is lower than a string to compare, -1 if the class string greater than a string to compare.

Left

Gets a specified number of characters from the left side of a string.

```
string  Left(
   uint  count      // Number of characters
   )
```

Parameters

count

[in] Number of characters.

Returned value

Resulted substring.

Right

Gets a specified number of characters from the right side of a string.

```
string  Right(
   uint  count      // Number of characters
   )
```

Parameters

count

[in] Number of characters.

Returned value

Resulted substring.

Mid

Gets a specified number of characters from a string.

```
string  Mid(
   uint  pos,        // Position
   uint  count       // Number of characters
   )
```

Parameters

pos

[in] Position of a string.

count

[in] Number of characters.

Returned value

Resulted substring.

Trim

Removes all leading and trailing occurrences of a set of specified characters (and ' ','\t','\r','\n') from a string.

```
int  Trim(
   const string  targets      // Set of characters to remove
   )
```

Parameters

targets

[in] Set of characters to remove.

Returned value

Number of characters removed.

Example:

```
//--- example for CString::Trim
#include <Strings\String.mqh>
//---
void OnStart()
  {
   CString str;
   //---
   str.Assign("   \t\tABCD\r\n");
   printf("Source string '%s'",str.Str());
   //---
   str.Trim("DA-DA-DA");
   printf("Result string '%s'",str.Str());
  }
```

TrimLeft

Removes all leading occurrences of a set of specified characters (and ' ','\t','\r','\n') from a string.

```
int  TrimLeft(
   const string  targets      // Set of characters to remove
   )
```

Parameters

targets

[in] Set of characters to remove.

Returned value

Number of characters removed.

TrimRight

Removes all trailing occurrences of a set of specified characters (and ' ','\t','\r','\n') from a string.

```
int  TrimRight(
   const string  targets      // Set of characters to remove
   )
```

Parameters

targets

[in] Set of characters to remove.

Returned value

Number of characters removed.

Clear

Clears a string.

```
bool  Clear()
```

Returned value

true if successful, false if a string hasn't been cleared.

ToUpper

Converts a string to uppercase.

```
bool  ToUpper()
```

Returned value

true if successful, false if a string hasn't been converted.

ToLower

Converts a string to lowercase.

```
bool  ToLower()
```

Returned value

true if successful, false if a string hasn't been converted.

Reverse

Reverses of a string.

```
void  Reverse()
```

Find

Searches for the first match of a substring.

```
int  Find(
   uint          start,          // Position
   const string  substring       // Substring to search for
   ) const;
```

Parameters

start

[in] The index of the character in the string to begin the search with, or 0 to start from the beginning.

substring

[in] Substring to search for.

Returned value

The index of the first character that matches the requested substring; -1 if the substring is not found.

FindRev

Searches for the last match of a substring.

```
int  FindRev(
   const string  substring      // Substring
   ) const;
```

Parameters

substring

[in] A substring to search for.

Returned value

The index of the last character that matches the requested substring; -1 if the substring is not found.

Remove

Deletes a substring from a string.

```
uint  Remove(
   const string  substring      // Substring to remove
   )
```

Parameters

substring

[in] A substring to search for.

Returned value

Number of substrings deleted.

Replace

Replaces a substring from a string.

```
uint  Replace(
   const string  substring,     // Substring to replace
   const string  newstring      // New substring
   )
```

Parameters

substring

[in] A substring to search for.

newstring

[in] A substring to replace for.

Returned value

Number of substrings replaced.

Technical Indicators and Timeseries

This section contains the technical details of the technical indicator and timeseries classes and description of the corresponding components of the standard MQL5 library.

The use of the technical indicator and timeseries classes will save time in developing applications (scripts, Expert Advisors).

The MQL5 Standard Library is placed in the working directory of the terminal in the Include\Indicators folder.

Class/group	Description
Base classes	Group of base and auxiliary classes
Timeseries classes	Group of timeseries classes
Trend Indicators	Group of Trend indicator classes
Oscillators	Group of Oscillator indicator classes
Volume Indicators	Group of Volume indicator classes
Bill Williams Indicators	Bill Williams indicator classes
Custom indicators	Custom indicator class

Base and Auxiliary Technical Indicator and Timeseries Classes

This section contains the technical details of base and auxiliary technical indicator and timeseries classes and description of the corresponding components of the Standard MQL5 library.

Class/group	Description
CSpreadBuffer	Historical spread buffer class
CTimeBuffer	Historical opening prices buffer class
CTickVolumeBuffer	Historical tick volumes buffer class
CRealVolumeBuffer	Historical real volumes buffer class
CDoubleBuffer	Base class of double type data buffer
COpenBuffer	Opening bar prices buffer class
CHighBuffer	High bar prices buffer class
CLowBuffer	Low bar prices buffer class
CCloseBuffer	Closing bar prices buffer class
CIndicatorBuffer	Technical indicator buffer class
CSeries	Base class for access to timeseries data
CPriceSeries	Base class for access to price data
CIndicator	Base class of technical indicator
CIndicators	Technical indicator and series collection

CSpreadBuffer

CSpreadBuffer is a class for simplified access to spreads of the bars in the history.

Description

The CSpreadBuffer class provides an access to spreads of the bars in the history.

Declaration

```
class CSpreadBuffer: public CArrayInt
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access Methods	
At	Gets the buffer element by index
Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Size

Sets buffer size.

```
void  Size(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element by index.

```
long  At(
   int  index      // index
   )
```

Parameters

index

[in] Index of buffer element.

Returned value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

CTimeBuffer

CTimeBuffer is a class for simplified access to opening times of the bars in the history.

Description

The CTimeBuffer class provides an access to opening times of the bars in the history.

Declaration

```
class CTimeBuffer: public CArrayLong
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access Methods	
At	Gets the buffer element by index
Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Size

Sets buffer size.

```
void  Size(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element by index.

```
long  At(
   int  index      // index
   )
```

Parameters

index

[in] Index of buffer element.

Returned value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

CTickVolumeBuffer

CTickVolumeBuffer is a class for simplified access to tick volumes of bars in the history.

Description

The CTickVolumeBuffer class provides an access to tick volumes of bars in the history.

Declaration

```
class CTickVolumeBuffer: public CArrayLong
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access Methods	
At	Gets the buffer element by index
Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Size

Sets buffer size.

```
void  Size(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element by index.

```
long  At(
   int  index      // index
   )
```

Parameters

index

[in] Index of buffer element.

Returned value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

CRealVolumeBuffer

CRealVolumeBuffer is a class for simplified access to real volumes of bars in the history.

Description

The CTickVolumeBuffer class provides an access to real volumes of bars in the history.

Declaration

```
class CRealVolumeBuffer: public CArrayLong
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access Methods	
At	Gets the buffer element by index
Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Size

Sets buffer size.

```
void  Size(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element by index.

```
long  At(
   int  index      // index
   )
```

Parameters

index

[in] Index of buffer element.

Returned value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

CDoubleBuffer

CDoubleBuffer is a base class for simplified access to data buffers of double type.

Description

The CDoubleBuffer class provides an access to the data of the buffer of double type.

Declaration

```
class CDoubleBuffer: public CArrayDouble
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Attributes	
Size	Sets buffer size
Settings	
SetSymbolPeriod	Sets symbol and period
Data Access Methods	
At	Gets the buffer element
Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Size

Sets buffer size.

```
void  Size(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

SetSymbolPeriod

Sets symbol and period.

```
void  SetSymbolPeriod(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] New symbol.

period

[in] New period (ENUM_TIMEFRAMES enumeration).

At

Gets the buffer element by index.

```
long  At(
   int  index      // index
   )
```

Parameters

index

[in] Index of buffer element.

Returned value

Buffer element with the specified index.

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

COpenBuffer

COpenBuffer is a class for simplified access to open prices of bars in the history.

Description

The COpenBuffer class provides an access to open prices of bars in the history.

Declaration

```
class COpenBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

CHighBuffer

CHighBuffer is a class for simplified access to high prices of bars in the history.

Description

The CHighBuffer class provides an access to high prices of bars in the history.

Declaration

```
class CHighBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

CLowBuffer

CLowBuffer is a class for simplified access to low prices of bars in the history.

Description

The CLowBuffer class provides an access to low prices of bars in the history.

Declaration

```
class CLowBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

CCloseBuffer

CCloseBuffer is a class for simplified access to close prices of bars in the history.

Description

The CCloseBuffer class provides an access to close prices of bars in the history.

Declaration

```
class CCloseBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Data Update Methods	
virtual Refresh	Updates the buffer
virtual RefreshCurrent	Updates the current value

Refresh

Updates the buffer.

```
virtual bool  Refresh()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) element of the buffer.

```
virtual bool  RefreshCurrent()
```

Returned value

true - if successful, false - if the buffer hasn't been updated.

CIndicatorBuffer

CIndicatorBuffer is a class for simplified access to the data of the indicator's buffer.

Description

The CIndicatorBuffer class provides the simplified access to the data buffer of technical indicator.

Declaration

```
class CIndicatorBuffer: public CDoubleBuffer
```

Title

```
#include <Indicators\Indicator.mqh>
```

Class Methods

Attributes	
Offset	Gets/Sets offset of the buffer
Name	Gets/Sets buffer name
Data Access Methods	
At	Gets buffer's element
Data Update Methods	
Refresh	Updates the buffer
RefreshCurrent	Updates only current value

Offset

Gets offset of the buffer.

```
int  Offset() const
```

Returned value

Buffer offset.

Offset ()

Sets offset of the buffer.

```
void  Offset(
   int  offset      // offset
   )
```

Parameters

offset

[in] New buffer offset.

Name

Gets the name of the buffer.

```
string  Name() const
```

Returned value

Name of the buffer.

Name

Sets the name of the buffer.

```
void  Name(
   string  name      // name
   )
```

Parameters

name

[in] New name of the buffer.

At

Gets buffer element by index.

```
long  At(
   int  index      // index
   )
```

Parameters

index

[in] Index of buffer element.

Returned value

Buffer element with the specified index.

Refresh

Updates the whole buffer.

```
bool  Refresh(
   int  handle,      // handle
   int  num          // buffer number
   )
```

Parameters

handle

[in] Handle of the indicator.

num

[in] Buffer index of the indicator.

Returned value

true - if successful, false - if the buffer hasn't been updated.

RefreshCurrent

Updates the current (zeroth) buffer element.

```
bool  RefreshCurrent(
   int  handle,     // handle of the indicator
   int  num         // buffer number
   )
```

Parameters

handle

[in] Handle of the indicator.

num

[in] Buffer number.

Returned value

true - if successful, false - if the buffer hasn't been updated.

CSeries

CSeries is a base class for an access to the timeseries data of the Standard Library.

Description

The CSeries class provides the simplified access to MQL5 timeseries functions to all its descendants.

Declaration

```
class CSeries: public CArrayObj
```

Title

```
#include <Indicators\Series.mqh>
```

Class Methods

Attributes	
Name	Gets the name of timeseries or indicator
BuffersTotal	Gets the number of buffers of timeseries or indicator
Timeframe	Gets the timeframe flag of timeseries or indicator
Symbol	Gets the symbol of timeseries or indicator
Period	Gets the period of timeseries or indicator
RefreshCurrent	Gets/Sets the flag of updating the current data
Data Access Methods	
virtual BufferResize	Sets buffer size of timeseries or indicator
Data Update Methods	
virtual Refresh	Update the data of timeseries or indicator
PeriodDescription	Gets the period as a string

Name

Gets the name of timeseries or indicator

```
string  Name() const
```

Returned value

The name of timeseries or indicator.

BuffersTotal

Gets the number of buffers of timeseries or indicator.

```
int  BuffersTotal() const
```

Returned value

The number of buffers of timeseries or indicator.

Note

The timeseries has an only one buffer.

Timeframe

Gets the timeframe flag of timeseries or indicator.

```
int  Timeframe() const
```

Returned value

The timeframe flag of timeseries or indicator.

Note

It's the visibility flag of some timeframes.

Symbol

Gets the symbol of timeseries or indicator.

```
string  Symbol() const
```

Returned value

The symbol of timeseries or indicator.

Period

Gets the period of timeseries or indicator.

```
ENUM_TIMEFRAMES  Period() const
```

Returned value

The period (value of ENUM_TIMEFRAMES enumeration) of timeseries or indicator.

RefreshCurrent

Sets a flag to update the current values of timeseries or indicator.

```
string  RefreshCurrent(
   bool  flag      // new flag
   )
```

Parameters

flag

[in] New flag.

Returned value

None.

BufferResize

Sets buffer size of timeseries or indicator.

```
virtual void  BufferResize(
   int  size      // new size
   )
```

Parameters

size

[in] New size of the buffers.

Note

All the buffers have the same buffer size.

Refresh

Updates the data of timeseries or indicator.

```
virtual void  Refresh(
   int  flags      // flags
   )
```

Parameters

flags

[in] Timeframes to update (flag).

PeriodDescription

Gets the string representation of the specified ENUM_TIMEFRAMES enumeration.

```
string  PeriodDescription(
   int  val=0      // value
   ) const
```

Parameters

val=0

[in] Value to convert.

Returned value

The string representation of the specified ENUM_TIMEFRAMES enumeration.

Note

If the value isn't specified or equal to zero, it returns the timeframe of timeseries or indicator.

CPriceSeries

CPriceSeries is a base class for access to the price data.

Description

The CSeries class provides the simplified access to MQL5 functions for working with price data to all its descendants.

Declaration

```
class CPriceSeries: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Create Methods	
virtual BufferResize	Sets size of the buffer
Data Access Methods	
virtual GetData	Gets the specified buffer element by index
Data Update Methods	
virtual Refresh	Updates timeseries data
Data Search Methods	
virtual MinIndex	Gets the index of minimal element in the specified range
virtual MinValue	Gets the value and index of minimal element in the specified range
virtual MaxIndex	Gets the index of maximal element in the specified range
virtual MaxValue	Gets the value and index of maximal element in the specified range

BufferResize

Sets new size of the buffer.

```
virtual void  BufferResize(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

GetData

Gets the specified buffer element by index.

```
virtual double  GetData(
   int  index      // index
   ) const
```

Parameters

index

[in] Index of buffer element.

Returned value

The buffer element with the specified index or EMPTY_VALUE.

Refresh

Updates the timeseries data

```
virtual void  Refresh(
   int  flags      // timeframe flags
   )
```

Parameters

flags

[in] Timeframes to update (flag).

MinIndex

Gets the index of minimal element in the specified range.

```
virtual int  MinIndex(
   int  start,      // starting index
   int  count       // number of elements to scan
   ) const
```

Parameters

start

[in] Starting index.

count

[in] Number of elements to proceed.

Returned value

The index of minimal element in the specified range, or -1 in the case of error.

MinValue

Gets the value and index of minimal element in the specified range.

```
virtual double  MinValue(
   int   start,      // starting index
   int   count,      // number of elements to scan
   int&  index       // reference to the variable for index
   ) const
```

Parameters

start

[in] Starting index.

count

[in] Number of elements to proceed.

index

[out] Reference to the variable of integer type.

Returned value

The value of minimal element of the buffer in the specified range, or EMPTY_VALUE if error.

Note

The index of minimal element is stored in the variable index.

MaxIndex

Gets the index of maximal element in the specified range.

```
virtual int  MaxIndex(
   int  start,      // starting index
   int  count       // number of elements to scan
   ) const
```

Parameters

start

[in] Starting index.

count

[in] Number of elements to proceed.

Returned value

The index of the maximal element in the specified range, or -1 in the case of error.

MaxValue

Gets the value and index of maximal element in the specified range.

```
virtual double  MaxValue(
   int   start,      // starting index
   int   count,      // number of elements to scan
   int&  index       // reference to the variable for index
   ) const
```

Parameters

start

[in] Starting index.

count

[in] Number of elements to proceed.

index

[out] Reference to the variable of integer type.

Returned value

The value of maximal element of the buffer in the specified range, or EMPTY_VALUE if error.

Note

The index of maximal element is stored in the variable index.

CIndicator

CIndicator is a base class for technical indicator classes of the standard MQL Library.

Description

The CIndicator class provides the simplified access for all of its descendants to MQL5 API technical indicator functions.

Declaration

```
class CIndicator: public CSeries
```

Title

```
#include <Indicators\Indicator.mqh>
```

Class Methods

Attributes	
Handle	Gets the indicator's handle.
Status	Gets the status of the indicator.
FullRelease	Sets a flag to release the handle of the indicator.
Creation	
Create	Creates the indicators
BufferResize	Sets new buffer size
Data Access Methods	
GetData	Copying the data from the indicator's buffer
Data Update Methods	
Refresh	Updates the indicator's data
Finding Min/Max Values	
Minimum	Gets the index of minimal element of the specified buffer in a specified range.
MinValue	Gets the value and index of minimal element of the specified buffer in a specified range.
Maximum	Gets the index of maximal element of the specified buffer in a specified range.
MaxValue	Gets the value and index of maximal element of the specified buffer in a specified range.
Conversion of Enumerations	

MethodDescription	Gets the value of ENUM_MA_METHOD enumeration as string
PriceDescription	Gets the value of ENUM_APPLIED_PRICE enumeration as string
VolumeDescription	Gets the value of ENUM_APPLIED_VOLUME enumeration as string
Working with chart	
AddToChart	Adds the indicator to the chart
DeleteFromChart	Deletes the indicator from the chart

Derived classes:

- CiAC
- CiAD
- CiADX
- CiADXWilder
- CiAlligator
- CiAMA
- CiAO
- CiATR
- CiBands
- CiBearsPower
- CiBullsPower
- CiBWMFI
- CiCCI
- CiChaikin
- CiDEMA
- CiDeMarker
- CiEnvelopes
- CiForce
- CiFractals
- CiFrAMA
- CiGator
- CiIchimoku
- CiMA
- CiMACD
- CiMFI
- CiMomentum
- CiOBV

- CiOsMA
- CiRSI
- CiRVI
- CiSAR
- CiStdDev
- CiStochastic
- CiTEMA
- CiTriX
- CiVIDyA
- CiVolumes
- CiWPR

Handle

Gets the indicator's handle.

```
int  Handle() const
```

Returned value

Handle of the indicator.

Status

Gets the status of the indicator.

```
string  Status() const
```

Returned value

The status of indicator creation.

FullRelease

Sets a flag to release the handle of the indicator.

```
void  FullRelease(
   bool  flag      // flag
   )
```

Parameters

flag

[in] New value of the handle release flag.

Create

Creates the indicator with specified parameters.

```
bool  Create(
   string            symbol,          // symbol
   ENUM_TIMEFRAMES   period,          // period
   ENUM_INDICATOR    type,            // type
   int               num_params,      // number of parameters
   MqlParam&         params           // reference to the parameters array
   )
```

Parameters

symbol

[in] Symbol name.

period

[in] Period (ENUM_TIMEFRAMES enumeration).

type

[in] Indicator's type (ENUM_INDICATOR enumeration).

num_params

[in] Number of indicator's parameters.

params

[in] Reference to the parameters array for the indicator.

Returned value

true if successful, false if indicator hasn't been created.

BufferResize

Sets the size of the indicator's buffer.

```
void  BufferResize(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

Note

All the indicator's buffers have the same size.

BarsCalculated

Returns the number of calculated data for the indicator.

```
int  BarsCalculated();
```

Return Value

Returns the amount of calculated data in the indicator buffer or -1 in the case of error (data not calculated yet).

GetData

Gets the specified element from the specified buffer of the indicator.

```
double  GetData(
   int  buffer_num,     // buffer number
   int  index           // element index
   ) const
```

Parameters

buffer_num

[in] Buffer number.

index

[in] Element index.

Returned value

If successful, it returns the numerical value of element, or EMPTY_VALUE in the case of error.

GetData

Gets the data from the indicator's buffer by starting position and number of necessary data.

```
int  GetData(
   int      start_pos,      // position
   int      count,          // number of elements needed
   int      buffer_num,     // buffer number
   double&  buffer          // target array for data
   ) const
```

Parameters

start_pos

[in] Starting position of the indicator's buffer.

count

[in] Number of elements needed.

buffer_num

[in] Number of the indicator's buffer.

buffer

[in] Reference to the target array for the data.

Returned value

If successful, it returns the number of elements, received from the specified indicator buffer, otherwise -1.

GetData

Gets the data from the indicator's buffer by start time and number of necessary data.

```
int  GetData(
   datetime  start_time,     // starting time
   int       count,          // number of elements needed
   int       buffer_num,     // buffer number
   double&   buffer          // target array for data
   ) const
```

Parameters

start_time

[in] Starting time.

count

[in] Number of elements needed.

buffer_num

[in] Number of the indicator's buffer.

buffer

[in] Reference to the target array.

Returned value

If successful, it returns the number of elements, received from the specified indicator buffer, otherwise -1.

GetData

Gets the data from the indicator's buffer by start and stop time and number of necessary data.

```
int  GetData(
   datetime  start_time,     // start time
   datetime  stop_time,      // stop time
   int       buffer_num,     // number of buffer
   double&   buffer          // target array for data
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer_num

[in] Number of the indicator's buffer.

buffer

[in] Reference to the target array.

Returned value

If successful, it returns the number of elements, received from the specified indicator buffer, otherwise -1.

Refresh

Updates the indicator's data.

```
void  Refresh(
   int  flags      // flags
   )
```

Parameters

flags

[in] Timeframe update flags.

Minimum

Returns the index of minimal element of the specified buffer in a specified range.

```
int  Minimum(
   int  buffer_num,     // buffer number
   int  start,          // starting index
   int  count           // number of elements to proceed
   ) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Starting index of the search.

count

[in] Number of elements to search.

Returned value

Index of the minimal element of the specified buffer in a specified range.

MinValue

Returns the value and index of minimal element of the specified buffer in a specified range.

```
double  MinValue(
   int   buffer_num,      // buffer number
   int   start,           // starting index
   int   count,           // number of elements to proceed
   int&  index            // reference
   ) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Starting index.

count

[in] Number of elements to proceed.

index

[out] Reference to the variable of int type for the minimal element index.

Returned value

The value of the minimal element of the specified buffer in a specified range.

Note

The index of minimal buffer element is stored into the variable index, passed by reference.

Maximum

Returns the index of maximal element of the specified buffer in a specified range.

```
int  Maximum(
   int  buffer_num,      // buffer number
   int  start,           // starting index
   int  count            // number of elements to proceed
   ) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Starting index of the search.

count

[in] Number of elements to search.

Returned value

Index of the maximal element of the specified buffer in a specified range.

MaxValue

Returns the value and index of maximal element of the specified buffer in a specified range.

```
double  MaxValue(
   int   buffer_num,      // buffer number
   int   start,           // starting index
   int   count,           // number of elements to proceed
   int&  index            // reference
   ) const
```

Parameters

buffer_num

[in] Buffer number to search the value in.

start

[in] Starting index.

count

[in] Number of elements to proceed.

index

[out] Reference to the variable of int type for the maximal element index.

Returned value

The value of the maximal element of the specified buffer in a specified range.

Note

The index of maximal buffer element is stored into the variable index, passed by reference.

MethodDescription

The function returns the value of ENUM_MA_METHOD enumeration as a string.

```
string  MethodDescription(
   int  val==0      // value
   ) const
```

Parameters

val==0

[in] Value of ENUM_MA_METHOD enumeration.

Returned value

The value of ENUM_MA_METHOD enumeration as a string.

PriceDescription

The method returns the value of ENUM_APPLIED_PRICE enumeration as a string.

```
string  PriceDescription(
   int  val==0      // value
   ) const
```

Parameters

val==0

[in] Value of ENUM_APPLIED_PRICE enumeration.

Returned value

The value of ENUM_APPLIED_PRICE enumeration as a string.

VolumeDescription

The method returns the value of ENUM_APPLIED_VOLUME enumeration as a string.

```
string  VolumeDescription(
   int  val==0      // value
   ) const
```

Parameters

val==0

[in] Value of ENUM_APPLIED_VOLUME enumeration.

Returned value

The value of ENUM_APPLIED_VOLUME enumeration as a string.

AddToChart

Adds the indicator to the chart.

```
bool  AddToChart(
   long  chart,      // chart ID
   int  subwin       // chart subwindow
   )
```

Parameters

chart

[in] Chart ID.

subwin

[in] Chart subwindow.

Returned value

true - if successful, false if error.

DeleteFromChart

Deletes the indicator from the chart.

```
bool  DeleteFromChart(
   long  chart,      // chart ID
   int  subwin       // chart subwindow
   )
```

Parameters

chart

[in] Chart ID.

subwin

[in] Chart subwindow.

Returned value

true - if successful, false if error.

CIndicators

The CIndicators is a class for collecting instances of timeseries and technical indicators classes.

Description

The CIndicators class provides creation of the technical indicators class instances, their storage and management (data synchronization, handle and memory management).

Declaration

```
class CIndicators: public CArrayObj
```

Title

```
#include <Indicators\Indicators.mqh>
```

Class Methods

Create Methods	
Create	Creates technical indicator
Data Update Methods	
Refresh	Updates data for all technical indicators in the collection

Create

It creates the indicator with the specified parameters.

```
CIndicator*  Create(
   string           symbol,     // Symbol name
   ENUM_TIMEFRAMES  period,     // Period
   ENUM_INDICATOR   type,       // Indicator's type
   int              count,      // Number of parameters
   MqlParam&        params      // Parameters array reference
   )
```

Parameters

symbol

[in] Symbol name.

period

[in] Period (ENUM_TIMEFRAMES enumeration).

type

[in] Indicator's type (ENUM_INDICATOR).

count

[in] Number of parameters for the indicator.

params

[in] Reference to the parameters array for the indicator.

Returned value

If successful, it returns the reference to the created indicator, and NULL if the indicator hasn't been created.

Refresh

Updates data for all technical indicators in the collection.

```
int  Refresh()
```

Returned value

It returns the updated timeframe flags (formed as an object visibility flags).

Timeseries classes

This group of chapters contains technical details of timeseries classes of the MQL5 Standard Library and descriptions of all its key components.

Class	Description
CiSpread	Provides an access to spread historical data
CiTime	Provides an access to open times of the bars in the history
CiTickVolume	Provides an access to tick volumes of the bars in the history
CiRealVolume	Provides an access to real volumes of the bars in the history
CiOpen	Provides an access to open prices of the bars in the history
CiHigh	Provides an access to high prices of the bars in the history
CiLow	Provides an access to low prices of the bars in the history
CiClose	Provides an access to close prices of the bars in the history

CiSpread

CiSpread is a class designed for access to spreads of the bars in the history.

Description

The CiSpread class provides an access to spread historical data.

Declaration

```
class CiSpread: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Create Methods	
Create	Creates a timeseries
BufferResize	Sets buffer size
Data Access Methods	
GetData	Gets the data
Data Update Methods	
Refresh	Updates the data

Create

Creates a timeseries with the specified parameters for access to the spreads history.

```
bool  Create(
   string           symbol,     // symbol
   ENUM_TIMEFRAMES  period      // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if timeseries hasn't been created.

BufferResize

Sets new size of the series.

```
virtual void  BufferResize(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

GetData

Gets the element of timeseries by index.

```
int  GetData(
   int  index      // index
   ) const
```

Parameters

index

[in] Index of the element needed.

Returned value

The timeseries buffer element, or 0.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int   start_pos,     // starting position
   int   count,         // number of elements to get
   int&  buffer         // target array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for the data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting time and number of elements.

```
int  GetData(
   datetime  start_time,     // starting time
   int       count,          // number of elements
   int&      buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(
   datetime  start_time,     // starting time
   datetime  stop_time,      // stop time
   int&      buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Returned value

>=0 if successful, -1 in the case of error.

Refresh

Updates the data of timeseries.

```
virtual void  Refresh(
   int  flags      // flags
   )
```

Parameters

flags

[in] Timeframe flags.

CiTime

CiTime is a class designed for access to open times of the bars in the history.

Description

The CiTime class provides an access to open times of the bars in the history.

Declaration

```
class CiTime: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Create Methods	
Create	Creates a timeseries
BufferResize	Sets buffer size
Data Access Methods	
GetData	Gets the data
Data Update Methods	
Refresh	Updates the data

Create

Creates a timeseries with the specified parameters for access to the opening times of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if timeseries hasn't been created.

BufferResize

Sets new size of the series.

```
virtual void  BufferResize(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

GetData

Gets the element of timeseries by index.

```
datetime  GetData(
   int  index      // index
   ) const
```

Parameters

index

[in] Index of the element needed.

Returned value

The timeseries buffer element, or 0.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int   start_pos,     // starting position
   int   count,         // number of elements to get
   long& buffer         // target array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for the data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting time and number of elements.

```
int  GetData(
   datetime  start_time,     // starting time
   int       count,          // number of elements
   long&     buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(
   datetime  start_time,     // starting time
   datetime  stop_time,      // stop time
   long&     buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Returned value

>=0 if successful, -1 in the case of error.

Refresh

Updates the data of timeseries.

```
virtual void  Refresh(
   int  flags      // flags
   )
```

Parameters

flags

[in] Timeframe flags.

CiTickVolume

CiTickVolume is a class designed for access to tick volumes of the bars in the history.

Description

The CiTickVolume class provides an access to tick volumes of the bars in the history.

Declaration

```
class CiTickVolume: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Create Methods	
Create	Creates a timeseries
BufferResize	Sets buffer size
Data Access Methods	
GetData	Gets the data
Data Update Methods	
Refresh	Updates the data

Create

Creates a timeseries with the specified parameters for access to the tick volumes of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if timeseries hasn't been created.

BufferResize

Sets new size of the series.

```
virtual void  BufferResize(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

GetData

Gets the element of timeseries by index.

```
datetime  GetData(
   int  index      // index
   ) const
```

Parameters

index

[in] Index of the element needed.

Returned value

The timeseries buffer element, or 0.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int   start_pos,     // starting position
   int   count,         // number of elements to get
   long& buffer         // target array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for the data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting time and number of elements.

```
int  GetData(
   datetime  start_time,     // starting time
   int       count,          // number of elements
   long&     buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(
   datetime  start_time,     // starting time
   datetime  stop_time,      // stop time
   long&     buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Returned value

>=0 if successful, -1 in the case of error.

Refresh

Updates the data of timeseries.

```
virtual void  Refresh(
   int  flags      // flags
   )
```

Parameters

flags

[in] Timeframe flags.

CiRealVolume

CiRealVolume is a class designed for access to real volumes of the bars in the history.

Description

The CiRealVolume class provides an access to real volumes of the bars in the history.

Declaration

```
class CiRealVolume: public CSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Create Methods	
Create	Creates a timeseries
BufferResize	Sets buffer size
Data Access Methods	
GetData	Gets the data
Data Update Methods	
Refresh	Updates the data

Create

Creates a timeseries with the specified parameters for access to the real volumes of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if timeseries hasn't been created.

BufferResize

Sets new size of the series.

```
virtual void  BufferResize(
   int  size      // new size
   )
```

Parameters

size

[in] New buffer size.

GetData

Gets the element of timeseries by index.

```
datetime  GetData(
   int  index      // index
   ) const
```

Parameters

index

[in] Index of the element needed.

Returned value

The timeseries buffer element, or 0.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int   start_pos,     // starting position
   int   count,         // number of elements to get
   long& buffer         // target array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for the data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting time and number of elements.

```
int  GetData(
   datetime  start_time,     // starting time
   int       count,          // number of elements
   long&     buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(
   datetime  start_time,     // starting time
   datetime  stop_time,      // stop time
   long&     buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Returned value

>=0 if successful, -1 in the case of error.

Refresh

Updates the data of timeseries.

```
virtual void  Refresh(
   int  flags      // flags
   )
```

Parameters

flags

[in] Timeframe flags.

CiOpen

CiOpen is a class designed for access to open prices of the bars in the history.

Description

The CiOpen class provides an access to open prices of the bars in the history.

Declaration

```
class CiOpen: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Create Methods	
Create	Creates a timeseries
Data Access Methods	
GetData	Gets the data

Create

Creates a timeseries with the specified parameters for access to the open prices of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if timeseries hasn't been created.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int    start_pos,      // starting position
   int    count,          // number of elements to get
   double& buffer         // target array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for the data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting time and number of elements.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number of elements
   double&     buffer         // target array
   ) const
```

Parameters

start_time

[in] Starting time.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(
   datetime  start_time,     // starting time
   datetime  stop_time,      // stop time
   double&   buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Returned value

>=0 if successful, -1 in the case of error.

CiHigh

CiHigh is a class designed for access to high prices of the bars in the history.

Description

The CiHigh class provides an access to high prices of the bars in the history.

Declaration

```
class CiHigh: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Create Methods	
Create	Creates a timeseries
Data Access Methods	
GetData	Gets the data

Create

Creates a timeseries with the specified parameters for access to the high prices of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if timeseries hasn't been created.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int    start_pos,      // starting position
   int    count,          // number of elements to get
   double& buffer         // target array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for the data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting time and number of elements.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number of elements
   double&     buffer         // target array
   ) const
```

Parameters

start_time

[in] Starting time.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(
   datetime  start_time,     // starting time
   datetime  stop_time,      // stop time
   double&   buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Returned value

>=0 if successful, -1 in the case of error.

CiLow

CiLow is a class designed for access to low prices of the bars in the history.

Description

The CiLow class provides an access to low prices of the bars in the history.

Declaration

```
class CiLow: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Create Methods	
Create	Creates a timeseries
Data Access Methods	
GetData	Gets the data

Create

Creates a timeseries with the specified parameters for access to the low prices of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if timeseries hasn't been created.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int    start_pos,      // starting position
   int    count,          // number of elements to get
   double& buffer         // target array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for the data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting time and number of elements.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number of elements
   double&     buffer         // target array
   ) const
```

Parameters

start_time

[in] Starting time.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(
   datetime  start_time,     // starting time
   datetime  stop_time,      // stop time
   double&   buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Returned value

>=0 if successful, -1 in the case of error.

CiClose

CiClose is a class designed for access to close prices of the bars in the history.

Description

The CiClose class provides an access to close prices of the bars in the history.

Declaration

```
class CiClose: public CPriceSeries
```

Title

```
#include <Indicators\TimeSeries.mqh>
```

Class Methods

Create Methods	
Create	Creates a timeseries
Data Access Methods	
GetData	Gets the data

Create

Creates a timeseries with the specified parameters for access to the closing prices of the bars in the history.

```
bool  Create(
   string           symbol,      // symbol
   ENUM_TIMEFRAMES  period       // period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if timeseries hasn't been created.

GetData

Gets the element of timeseries by starting position and number of elements.

```
int  GetData(
   int    start_pos,      // starting position
   int    count,          // number of elements to get
   double& buffer         // target array
   ) const
```

Parameters

start_pos

[in] Starting position of timeseries.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for the data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting time and number of elements.

```
int  GetData(
   datetime  start_time,      // starting time
   int       count,           // number of elements
   double&     buffer         // target array
   ) const
```

Parameters

start_time

[in] Starting time.

count

[in] Number of elements needed.

buffer

[in] Reference to the target array for data.

Returned value

>=0 if successful, -1 in the case of error.

GetData

Gets the element of timeseries by starting and stop times.

```
int  GetData(
   datetime  start_time,     // starting time
   datetime  stop_time,      // stop time
   double&   buffer          // target array
   ) const
```

Parameters

start_time

[in] Starting time.

stop_time

[in] Stop time.

buffer

[in] Reference to the target array for data

Returned value

>=0 if successful, -1 in the case of error.

Trend Indicator Classes

This group of chapters contains technical details of Trend Indicator classes of the MQL5 Standard Library and descriptions of all its key components.

Class/group	Description
CiADX	Average Directional Index
CiADXWilder	Average Directional Index by Welles Wilder
CiBands	Bollinger Bands®
CiEnvelopes	Envelopes
CiIchimoku	Ichimoku Kinko Hyo
CiMA	Moving Average
CiSAR	Parabolic Stop And Reverse System
CiStdDev	Standard Deviation
CiDEMA	Double Exponential Moving Average
CiTEMA	Triple Exponential Moving Average
CiFrAMA	Fractal Adaptive Moving Average
CiAMA	Adaptive Moving Average
CiVIDyA	Variable Index DYnamic Average

CiADX

CiADX is a class intended for using the Average Directional Index technical indicator.

Description

The CiADX class provides the creation and access to the data of the Average Directional Index indicator.

Declaration

```
class CiADX: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element of the main line
Plus	Returns the buffer element of the +DI line
Minus	Returns the buffer element of the -DI line
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period      // Averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Plus

Returns the buffer element of the +DI line by the specified index.

```
double  Plus(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the +DI line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Minus

Returns the buffer element of the -DI line by the specified index.

```
double  Minus(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the -DI line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_ADX for CiADX).

CiADXWilder

CiADXWilder is a class intended for using the Average Directional Index by Welles Wilder technical indicator.

Description

The CiADXWilder class provides the creation and access to the data of the Average Directional Index by Welles Wilder indicator.

Declaration

```
class CiADXWilder: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element of the main line
Plus	Returns the buffer element of the +DI line
Minus	Returns the buffer element of the -DI line
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string            symbol,        // Symbol
   ENUM_TIMEFRAMES   period,        // Period
   int               ma_period      // Averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Plus

Returns the buffer element of the +DI line by the specified index.

```
double  Plus(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the +DI line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Minus

Returns the buffer element of the -DI line by the specified index.

```
double  Minus(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the -DI line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_ADXW for CiADXWilder).

CiBands

CiBands is a class intended for using the Bollinger Bands® technical indicator.

Description

The CiBands class provides the creation and access to the data of the Bollinger Bands indicator.

Declaration

```
class CiBands: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
Deviation	Returns the deviation
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element of the base line
Upper	Returns the buffer element of the upper line
Lower	Returns the buffer element of the lower line
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int  MaShift() const
```

Returned value

Returns the horizontal shift value, defined at the indicator creation.

Deviation

Returns the deviation.

```
double  Deviation() const
```

Returned value

Returns the deviation, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period,     // Averaging period
   int              ma_shift,      // Shift
   double           deviation,     // Deviation
   int              applied        // applied price, or handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift of the indicator.

deviation

[in] Deviation.

applied

[in] Volume type to apply.

Returned value

true if successful, false if indicator hasn't been created.

Base

Returns the buffer element of the base line by the specified index.

```
double  Base(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the Base line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Upper

Returns the buffer element of the upper line by the specified index.

```
double  Upper(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the upper line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Lower

Returns the buffer element of the lower line by the specified index.

```
double  Lower(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the lower line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_BANDS for CiBands).

CiEnvelopes

CiEnvelopes is a class intended for using the Envelopes technical indicator.

Description

The CiEnvelopes class provides the creation and access to the data of the Envelopes indicator.

Declaration

```
class CiEnvelopes: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
MaMethod	Returns the averaging method
Deviation	Returns the deviation
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Upper	Returns the buffer element of the upper line
Lower	Returns the buffer element of the lower line
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int  MaShift() const
```

Returned value

Returns the horizontal shift value, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Returned value

Returns the averaging method, defined at the indicator creation.

Deviation

Returns the value of deviation.

```
double  Deviation() const
```

Returned value

Returns the value of deviation, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string            symbol,        // Symbol
   ENUM_TIMEFRAMES   period,        // Period
   int               ma_period,     // Averaging period
   int               ma_shift,      // Horizontal shift
   ENUM_MA_METHOD    ma_method,     // Averaging method
   int               applied,       // Price type or handle to apply
   double            deviation      // Deviation
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration).

applied

[in] Price type of handle to apply.

deviation

[in] Deviation.

Returned value

true if successful, false if indicator hasn't been created.

Upper

Returns the buffer element of the upper line by the specified index.

```
double  Upper(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the upper line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Lower

Returns the buffer element of the lower line by the specified index.

```
double  Lower(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the lower line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_ENVELOPES for CiEnvelopes).

CiIchimoku

CiIchimoku is a class intended for using the Ichimoku Kinko Hyo technical indicator.

Description

The CiIchimoku class provides the creation, setup and access to the data of the Ichimoku Kinko Hyo indicator.

Declaration

```
class CiIchimoku: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
TenkanSenPeriod	Returns the TenkanSen period
KijunSenPeriod	Returns the KijunSen period
SenkouSpanBPeriod	Returns the SenkouSpanB period
Create Methods	
Create	Creates the indicator
Data Access Methods	
TenkanSen	Returns the buffer element of the TenkanSen line
KijunSen	Returns the buffer element of the KijunSen line
SenkouSpanA	Returns the buffer element of the SenkouSpanA line
SenkouSpanB	Returns the buffer element of the SenkouSpanB line
ChinkouSpan	Returns the buffer element of the ChinkouSpan line
Input/output	
virtual Type	Returns the object type identifier

TenkanSenPeriod

Returns the TenkanSen period.

```
int  TenkanSenPeriod() const
```

Returned value

Returns the TenkanSen period, defined at the indicator creation.

KijunSenPeriod

Returns the KijunSen period.

```
int  KijunSenPeriod() const
```

Returned value

Returns the KijunSen period, defined at the indicator creation.

SenkouSpanBPeriod

Returns the SenkouSpanB period.

```
int  SenkouSpanBPeriod() const
```

Returned value

Returns the SenkouSpanB period, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,           // Symbol
   ENUM_TIMEFRAMES  period,           // Period
   int              tenkan_sen,       // Period of TenkanSen
   int              kijun_sen,        // Period of KijunSen
   int              senkou_span_b     // Period of SenkouSpanB
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

tenkan_sen

[in] Period of TenkanSen.

kijun_sen

[in] Period of KijunSen.

senkou_span_b

[in] Period of SenkouSpanB.

Returned value

true if successful, false if indicator hasn't been created.

TenkanSen

Returns the buffer element of the TenkanSen line by the specified index.

```
double  TenkanSen(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the TenkanSen line of the specified index, or EMPTY_VALUE if there isn't any correct data.

KijunSen

Returns the buffer element of the KijunSen line by the specified index.

```
double  KijunSen(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the KijunSen line of the specified index, or EMPTY_VALUE if there isn't any correct data.

SenkouSpanA

Returns the buffer element of the SenkouSpanA line by the specified index.

```
double  SenkouSpanA(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the SenkouSpanA line of the specified index, or EMPTY_VALUE if there isn't any correct data.

SenkouSpanB

Returns the buffer element of the SenkouSpanB line by the specified index.

```
double  SenkouSpanB(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the SenkouSpanB line of the specified index, or EMPTY_VALUE if there isn't any correct data.

ChinkouSpan

Returns the buffer element of the ChinkouSpan line by the specified index.

```
double  ChinkouSpan(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the ChinkouSpan line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_ICHIMOKU for Cilchimoku).

CiMA

CiMA is a class intended for using the Moving Average technical indicator.

Description

The CiMA class provides the creation, setup and access to the data of the Moving Average indicator.

Declaration

```
class CiMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int  MaShift() const
```

Returned value

Returns the horizontal shift value, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Returned value

Returns the averaging method (value of ENUM_MA_METHOD enumeration), defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string            string,        // Symbol
   ENUM_TIMEFRAMES   period,        // Period
   int               ma_period,     // Averaging period
   int               ma_shift,      // Horizontal shift
   ENUM_MA_METHOD    ma_method,     // Averaging method
   int               applied        // Price type of handle to apply
   )
```

Parameters

string

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration).

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_MA for CiMA).

CiSAR

CiSAR is a class intended for using the Parabolic Stop And Reverse System technical indicator.

Description

The CiSAR class provides the creation, setup and access to the data of the Parabolic Stop And Reverse System indicator.

Declaration

```
class CiSAR: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
SarStep	Returns the step for the velocity increasing
Maximum	Returns the coefficient of price following
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

SarStep

Returns the step for the velocity increasing (acceleration coefficient).

```
double  SarStep() const
```

Returned value

The step for the velocity increasing, defined at the indicator creation.

Maximum

Returns the coefficient of price following.

```
double  Maximum() const
```

Returned value

The price following coefficient, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,      // Symbol
   ENUM_TIMEFRAMES  period,      // Period
   double           step,        // Step
   double           maximum      // Coefficient
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

step

[in] Step for the velocity increasing.

maximum

[in] Price following coefficient.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_SAR for CiSAR).

CiStdDev

CiStdDev is a class intended for using the Standard Deviation technical indicator.

Description

The CiStdDev class provides the creation, setup and access to the data of the Standard Deviation indicator.

Declaration

```
class CiStdDev: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
MaShift	Returns the horizontal shift
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

MaShift

Returns the horizontal shift of the indicator.

```
int  MaShift() const
```

Returned value

Returns the horizontal shift value, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Returned value

Returns the averaging method (value of ENUM_MA_METHOD enumeration), defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string            symbol,         // Symbol
   ENUM_TIMEFRAMES   period,         // Period
   int               ma_period,      // Averaging period
   int               ma_shift,       // Horizontal shift
   ENUM_MA_METHOD    ma_method,      // Averaging method
   int               applied         // Price type or handle to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration).

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_STDDEV for CiStdDev).

CiDEMA

CiDEMA is a class intended for using the Double Exponential Moving Average technical indicator.

Description

The CiDEMA class provides the creation, setup and access to the data of the Double Exponential Moving Average indicator.

Declaration

```
class CiDEMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Returned value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           string,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period,     // Averaging period
   int              ind_shift,     // Shift
   int              applied        // Price type of handle to apply
   )
```

Parameters

string

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

ind_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_DEMA for CiDEMA).

CiTEMA

CiTEMA is a class intended for using the Triple Exponential Moving Average technical indicator.

Description

The CiTEMA class provides the creation, setup and access to the data of the Triple Exponential Moving Average indicator.

Declaration

```
class CiTEMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Returned value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period,     // Averaging period
   int              ma_shift,      // Offset
   int              applied        // Price type of handle to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_TEMA for CiTEMA).

CiFrAMA

CiFrAMA is a class intended for using the Fractal Adaptive Moving Average technical indicator.

Description

The CiFrAMA class provides the creation, setup and access to the data of the Fractal Adaptive Moving Average indicator.

Declaration

```
class CiFrAMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Returned value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period,     // Averaging period
   int              ma_shift,      // Offset
   int              applied        // Price type of handle to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

ma_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_FRAMA for CiFrAMA).

CiAMA

CiAMA is a class intended for using the Adaptive Moving Average technical indicator.

Description

The CiAMA class provides the creation, setup and access to the data of the Adaptive Moving Average indicator.

Declaration

```
class CiAMA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
FastEmaPeriod	Returns the averaging period for the fast EMA
SlowEmaPeriod	Returns the averaging period for the slow EMA
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

FastEmaPeriod

Returns the averaging period for the fast EMA.

```
int  FastEmaPeriod() const
```

Returned value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowEmaPeriod

Returns the averaging period for the slow EMA.

```
int  SlowEmaPeriod() const
```

Returned value

Returns the averaging period for the slow EMA, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Returned value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           string,              // Symbol
   ENUM_TIMEFRAMES  period,              // Period
   int              ma_period,           // Averaging period
   int              fast_ema_period,     // Fast EMA period
   int              slow_ema_period,     // Slow EMA period
   int              ind_shift,           // Shift
   int              applied              // Price type or handle to apply
   )
```

Parameters

string

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

fast_ema_period

[in] Fast EMA averaging period.

slow_ema_period

[in] Slow EMA averaging period.

ind_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_AMA for CiAMA).

CiVIDyA

CiVIDyA is a class intended for using the Variable Index DYnamic Average technical indicator.

Description

The CiVIDyA class provides the creation, setup and access to the data of the Variable Index DYnamic Average indicator.

Declaration

```
class CiVIDyA: public CIndicator
```

Title

```
#include <Indicators\Trend.mqh>
```

Class Methods

Attributes	
CmoPeriod	Returns the period for Momentum
EmaPeriod	Returns the averaging period for EMA
IndShift	Returns the horizontal shift
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

CmoPeriod

Returns the period for Momentum.

```
int  CmoPeriod() const
```

Returned value

Returns the period for Momentum, defined at the indicator creation.

EmaPeriod

Returns the averaging period for EMA.

```
int  EmaPeriod() const
```

Returned value

Returns the averaging period for EMA, defined at the indicator creation.

IndShift

Returns the horizontal shift of the indicator.

```
int  IndShift() const
```

Returned value

Returns the horizontal shift value, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,          // Symbol
   ENUM_TIMEFRAMES  period,          // Period
   int              cmo_period,      // Momentum period
   int              ema_period,      // Averaging period
   int              ind_shift,       // Shift
   int              applied          // Price type or handle to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

cmo_period

[in] Momentum period.

ema_period

[in] Averaging period.

ind_shift

[in] Horizontal shift.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_VIDYA for CiVIDyA).

Oscillators

This group of chapters contains the technical details of Oscillators classes of the MQL5 Standard Library and descriptions of all its key components.

Class/group	Description
CiATR	Average True Range
CiBearsPower	Bears Power
CiBullsPower	Bulls Power
CiCCI	Commodity Channel Index
CiChaikin	Chaikin Oscillator
CiDeMarker	DeMarker
CiForce	Force Index
CiMACD	Moving Averages Convergence-Divergence
CiMomentum	Momentum
CiOsMA	Moving Average of Oscillator (MACD histogram)
CiRSI	Relative Strength Index
CiRVI	Relative Vigor Index
CiStochastic	Stochastic Oscillator
CiWPR	Williams' Percent Range
CiTriX	Triple Exponential Moving Averages Oscillator

CiATR

CiATR is a class intended for using the Average True Range technical indicator.

Description

The CiATR class provides the creation, setup and access to the data of the Average True Range indicator.

Declaration

```
class CiATR: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string            symbol,         // Symbol
   ENUM_TIMEFRAMES   period,         // Period
   int               ma_period       // Averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_ATR for CiATR).

CiBearsPower

CiBearsPower is a class intended for using the Bears Power technical indicator.

Description

The CiBearsPower class provides the creation, setup and access to the data of the Bears Power indicator.

Declaration

```
class CiBearsPower: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period      // Averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_BEARS for CiBearsPower).

CiBullsPower

CiBullsPower is a class intended for using the Bulls Power technical indicator.

Description

The CiBullsPower class provides the creation, setup and access to the data of the Bulls Power indicator.

Declaration

```
class CiBullsPower: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period      // Averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_BULLS for CiBullsPower).

CiCCI

CiCCI is a class intended for using the Commodity Channel Index technical indicator.

Description

The CiCCI class provides the creation, setup and access to the data of the Commodity Channel Index indicator.

Declaration

```
class CiCCI: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period,     // Averaging period
   int              applied        // Price type or handle to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_CCI for CiCCI).

CiChaikin

CiChaikin is a class intended for using the Chaikin Oscillator technical indicator.

Description

The CiChaikin class provides the creation, setup and access to the data of the Chaikin Oscillator indicator.

Declaration

```
class CiChaikin: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
FastMaPeriod	Returns the averaging period for the fast MA
SlowMaPeriod	Returns the averaging period for the slow MA
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

FastMaPeriod

Returns the averaging period for the fast EMA.

```
int  FastMaPeriod() const
```

Returned value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowMaPeriod

Returns the averaging period for the slow EMA.

```
int  SlowMaPeriod() const
```

Returned value

Returns the averaging period for the slow EMA, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Returned value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Returned value

Volume type to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string               symbol,              // Symbol
   ENUM_TIMEFRAMES      period,              // Period
   int                  fast_ma_period,      // Fast EMA period
   int                  slow_ma_period,      // Slow EMA period
   ENUM_MA_METHOD       ma_method,           // Averaging method
   ENUM_APPLIED_VOLUME  applied              // Volume type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

fast_ma_period

[in] Period for fast EMA.

slow_ma_period

[in] Period for slow EMA.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_CHAIKIN for CiChaikin).

CiDeMarker

CiDeMarker is a class intended for using the DeMarker technical indicator.

Description

The CiDeMarker class provides the creation, setup and access to the data of the DeMarker indicator.

Declaration

```
class CiDeMarker: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period      // Averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_DEMARKER for CiDeMarker).

CiForce

CiForce is a class intended for using the Force Index technical indicator.

Description

The CiForce class provides the creation, setup and access to the data of the Force Index indicator.

Declaration

```
class CiForce: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Returned value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Returned value

Volume type to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string               symbol,        // Symbol
   ENUM_TIMEFRAMES      period,        // Period
   int                  ma_period,     // Averaging period
   ENUM_MA_METHOD       ma_method,     // Averaging method
   ENUM_APPLIED_VOLUME  applied        // Volume type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_FORCE for CiForce).

CiMACD

CiMACD is a class intended for using the Moving Averages Convergence-Divergence technical indicator.

Description

The CiMACD class provides the creation, setup and access to the data of the Moving Averages Convergence-Divergence indicator.

Declaration

```
class CiMACD: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
FastEmaPeriod	Returns the averaging period fofor the fast EMA
SlowEmaPeriod	Returns the averaging period of the slow EMA
SignalPeriod	Returns the averaging period of the signal line
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element of the main line
Signal	Returns the buffer element of the signal line
Input/output	
virtual Type	Returns the object type identifier

FastEmaPeriod

Returns the averaging period for the fast EMA.

```
int  FastEmaPeriod() const
```

Returned value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowEmaPeriod

Returns the averaging period for the slow EMA.

```
int  SlowEmaPeriod() const
```

Returned value

Returns the averaging period for the slow EMA, defined at the indicator creation.

SignalPeriod

Returns the averaging period for the signal line.

```
int  SignalPeriod() const
```

Returned value

Returns the averaging period for the signal line, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,              // Symbol
   ENUM_TIMEFRAMES  period,              // Period
   int              fast_ema_period,     // Fast EMA period
   int              slow_ema_period,     // Slow EMA period
   int              signal_period,       // Signal period
   int              applied              // Price type or handle to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

fast_ema_period

[in] Fast EMA period.

slow_ema_period

[in] Slow EMA period.

signal_period

[in] Signal line period.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Signal

Returns the buffer element of the signal line by the specified index.

```
double  Signal(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the signal line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_MACD for CiMACD).

CiMomentum

CiMomentum is a class intended for using the Momentum technical indicator.

Description

The CiMomentum class provides the creation, setup and access to the data of the Momentum indicator.

Declaration

```
class CiMomentum: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period,     // Averaging period
   int              applied        // Price type or handle to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_MOMENTUM for CiMomentum).

CiOsMA

CiOsMA is a class intended for using the Moving Average of Oscillator (MACD histogram) technical indicator.

Description

The CiOsMA class provides the creation, setup and access to the data of the Moving Average of Oscillator (MACD histogram) indicator.

Declaration

```
class CiOsMA: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
FastEmaPeriod	Returns the averaging period of the fast EMA
SlowEmaPeriod	Returns the averaging period of the slow EMA
SignalPeriod	Returns the averaging period of the signal line
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

FastEmaPeriod

Returns the averaging period for the fast EMA.

```
int  FastEmaPeriod() const
```

Returned value

Returns the averaging period for the fast EMA, defined at the indicator creation.

SlowEmaPeriod

Returns the averaging period for the slow EMA.

```
int  SlowEmaPeriod() const
```

Returned value

Returns the averaging period for the slow EMA, defined at the indicator creation.

SignalPeriod

Returns the averaging period for the signal line.

```
int  SignalPeriod() const
```

Returned value

Returns the averaging period for the signal line, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,              // Symbol
   ENUM_TIMEFRAMES  period,              // Period
   int              fast_ema_period,     // Fast EMA period
   int              slow_ema_period,     // Slow EMA period
   int              signal_period,       // Signal line period
   int              applied              // Price type or handle to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

fast_ema_period

[in] Fast EMA period.

slow_ema_period

[in] Slow EMA period.

signal_period

[in] Signal line period.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_OSMA for CiOsMA).

CiRSI

CiRSI is a class intended for using the Relative Strength Index technical indicator.

Description

The CiRSI class provides the creation, setup and access to the data of the Relative Strength Index indicator.

Declaration

```
class CiRSI: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period,     // Averaging period
   int              applied        // Price type or handle to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

applied

[in] Price type or handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_RSI for CiRSI).

CiRVI

CiRVI is a class intended for using the Relative Vigor Index technical indicator.

Description

The CiRVI class provides the creation, setup and access to the data of the Relative Vigor Index indicator.

Declaration

```
class CiRVI: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element of the base line
Signal	Returns the buffer element of the signal line
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period      // Averaging period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Signal

Returns the buffer element of the signal line by the specified index.

```
double  Signal(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the signal line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_RVI for CiRVI).

CiStochastic

CiStochastic is a class intended for using the Stochastic Oscillator technical indicator.

Description

The CiStochastic class provides the creation, setup and access to the data of the Stochastic Oscillator indicator.

Declaration

```
class CiStochastic: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
Kperiod	Returns the averaging period for the %K line
Dperiod	Returns the averaging period for the %D line
Slowing	Returns the slowing period
MaMethod	Returns the averaging method
PriceField	Price type (Low/High ore Close/Close) to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element of the base line
Signal	Returns the buffer element of the signal line
Input/output	
virtual Type	Returns the object type identifier

Kperiod

Returns the averaging period for the %K line.

```
int  Kperiod() const
```

Returned value

Returns the averaging period for the %K line, defined at the indicator creation.

Dperiod

Returns the averaging period for the %D line.

```
int  Dperiod() const
```

Returned value

Returns the averaging period for the %D line, defined at the indicator creation.

Slowing

Returns the period of slowing.

```
int  Slowing() const
```

Returned value

Returns the period of slowing, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Returned value

Returns the averaging method, defined at the indicator creation.

PriceField

Returns the price type (Low/High or Close/Close) to apply.

```
ENUM_STO_PRICE  PriceField() const
```

Returned value

The price type to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string            symbol,          // Symbol
   ENUM_TIMEFRAMES   period,          // Period
   int               Kperiod,         // Averaging period of %K
   int               Dperiod,         // Averaging period of %D
   int               slowing,         // Slowing period
   ENUM_MA_METHOD    ma_method,       // Averaging method
   ENUM_STO_PRICE    price_field      // Price type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Kperiod

[in] Averaging period of %K line.

Dperiod

[in] Averaging period of %D line.

slowing

[in] Slowing period.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration).

price_field

[in] Price type (Low/High or Close/Close) to apply (ENUM_STO_PRICE enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Signal

Returns the buffer element of the signal line by the specified index.

```
double  Signal(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Signal

Returns the buffer element of the signal line by the specified index.

```
double  Signal(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the signal line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_STOCHASTIC for CiStochastic).

CiTriX

CiTriX is a class intended for using the Triple Exponential Moving Averages Oscillator technical indicator.

Description

The CiTriX class provides the creation, setup and access to the data of the Triple Exponential Moving Averages Oscillator indicator.

Declaration

```
class CiTriX: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,        // Symbol
   ENUM_TIMEFRAMES  period,        // Period
   int              ma_period,     // Averaging period
   int              applied        // Price type or handle
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

applied

[in] Price type of handle to apply.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_TRIX for CiTriX).

CiWPR

CiWPR is a class intended for using the Williams' Percent Range technical indicator.

Description

The CiWPR class provides the creation, setup and access to the data of the Williams' Percent Range indicator.

Declaration

```
class CiWPR: public CIndicator
```

Title

```
#include <Indicators\Oscilators.mqh>
```

Class Methods

Attributes	
CalcPeriod	Returns the calculation period
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

CalcPeriod

Returns the period for calculation.

```
int  CalcPeriod() const
```

Returned value

Returns the the period for calculation, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,          // Symbol
   ENUM_TIMEFRAMES  period,          // Period
   int              calc_period      // Calculation period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

calc_period

[in] Period for calculation.

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_WPR for CiWPR).

Volume Indicators

This group of chapters contains technical details of Volume Indicator classes of the MQL5 Standard Library and descriptions of all its key components.

Class/group	Description
CiAD	Accumulation/Distribution
CiMFI	Money Flow Index
CiOBV	On Balance Volume
CiVolumes	Volumes

CiAD

CiAD is a class intended for using the Accumulation/Distribution technical indicator.

Description

The CiAD class provides the creation, setup and access to the data of the Accumulation/Distribution indicator.

Declaration

```
class CiAD: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Class Methods

Attributes	
Applied	Returns the volume type to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Returned value

Volume type to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string                symbol,      // Symbol
   ENUM_TIMEFRAMES       period,      // Period
   ENUM_APPLIED_VOLUME   applied      // Volume type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_AD for CiAD).

CiMFI

CiMFI is a class intended for using the Money Flow Index technical indicator.

Description

The CiMFI class provides the creation, setup and access to the data of the Money Flow Index indicator.

Declaration

```
class CiMFI: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Class Methods

Attributes	
MaPeriod	Returns the averaging period
Applied	Returns the volume type to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

MaPeriod

Returns the averaging period.

```
int  MaPeriod() const
```

Returned value

Returns the averaging period, defined at the indicator creation.

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Returned value

Volume type to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string                symbol,         // Symbol
   ENUM_TIMEFRAMES       period,         // Period
   int                   ma_period,      // Averaging period
   ENUM_APPLIED_VOLUME   applied         // Volume type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

ma_period

[in] Averaging period.

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_MFI for CiMFI).

CiOBV

CiOBV is a class intended for using the On Balance Volume technical indicator.

Description

The CiOBV class provides the creation, setup and access to the data of the On Balance Volume indicator.

Declaration

```
class CiOBV: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Class Methods

Attributes	
Applied	Returns the volume type to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Returned value

Volume type to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string                symbol,      // Symbol
   ENUM_TIMEFRAMES       period,      // Period
   ENUM_APPLIED_VOLUME   applied      // Volume type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_OBV for CiOBV).

CiVolumes

CiVolumes is a class intended for using the Volumes technical indicator.

Description

The CiVolumes class provides the creation, setup and access to the data of the Volumes indicator.

Declaration

```
class CiVolumes: public CIndicator
```

Title

```
#include <Indicators\Volumes.mqh>
```

Class Methods

Attributes	
Applied	Returns the volume type to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Returned value

Volume type to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string                symbol,      // Symbol
   ENUM_TIMEFRAMES       period,      // Period
   ENUM_APPLIED_VOLUME   applied      // Volume type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_VOLUMES for CiVolumes).

Bill Williams Indicators

This group of chapters contains technical details of Bill Williams Indicator classes of the MQL5 Standard Library and descriptions of all its key components.

Class/group	Description
CiAC	Accelerator Oscillator
CiAlligator	Alligator
CiAO	Awesome Oscillator
CiFractals	Fractals
CiGator	Gator Oscillator
CiBWMFI	Market Facilitation Index

CiAC

CiAC is a class intended for using the Accelerator Oscillator technical indicator.

Description

The CiAC class provides the creation, setup and access to the data of the Accelerator Oscillator indicator.

Declaration

```
class CiAC: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Class Methods

Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,     // Symbol
   ENUM_TIMEFRAMES  period      // Period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_AC for CiAC).

CiAlligator

CiAlligator is a class intended for using the Alligator technical indicator.

Description

The CiAlligator class provides the creation, setup and access to the data of the Alligator indicator.

Declaration

```
class CiAlligator: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Class Methods

Attributes	
JawPeriod	Returns the averaging period for the Jaws line
JawShift	Returns the horizontal shift of the Jaws line
TeethPeriod	Returns the averaging period for the Teeths line
TeethShift	Returns the horizontal shift of the Teeths line
LipsPeriod	Returns the averaging period for the Lips line
LipsShift	Returns the horizontal shift of the Lips line
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Jaw	Returns the buffer element of the Jaws line buffer
Teeth	Returns the buffer element of the Teeths line buffer
Lips	Returns the buffer element of the Lips line buffer
Input/output	
virtual Type	Returns the object type identifier

JawPeriod

Returns the averaging period for the Jaw line.

```
int  JawPeriod() const
```

Returned value

Returns the averaging period for the Jaw line, defined at the indicator creation.

JawShift

Returns the horizontal shift of the Jaws line.

```
int  JawShift() const
```

Returned value

Horizontal shift of the Jaws line, defined at the indicator creation.

TeethPeriod

Returns the averaging period for the Teeth line.

```
int  TeethPeriod() const
```

Returned value

Returns the averaging period for the Teeth line, defined at the indicator creation.

TeethShift

Returns the horizontal shift of the Teeths line.

```
int  TeethShift() const
```

Returned value

Horizontal shift of the Teeths line, defined at the indicator creation.

LipsPeriod

Returns the averaging period for the Lips line.

```
int  LipsPeriod() const
```

Returned value

Returns the averaging period for the Lips line, defined at the indicator creation.

LipsShift

Returns the horizontal shift of the Lips line.

```
int  LipsShift() const
```

Returned value

Horizontal shift of the Lips line, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Returned value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string            symbol,           // Symbol
   ENUM_TIMEFRAMES   period,           // Period
   int               jaw_period,       // Jaws period
   int               jaw_shift,        // Jaws shift
   int               teeth_period,     // Teeths period
   int               teeth_shift,      // Teeths shift
   int               lips_period,      // Lips period
   int               lips_shift,       // Lips shift
   ENUM_MA_METHOD    ma_method,        // Averaging method
   int               applied           // Volume type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

jaw_period

[in] Jaws period.

jaw_shift

[in] Jaws shift.

teeth_period

[in] Teeths period.

teeth_shift

[in] Teeths shift.

lips_period

[in] Lips period.

lips_shift

[in] Lips shift.

ma_method

[in] Moving average method (ENUM_MA_METHOD enumeration).

applied

[in] Volume type to apply.

Returned value

true if successful, false if indicator hasn't been created.

Jaw

Returns the buffer element of the Jaws line by the specified index.

```
double  Jaw(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the Jaws line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Teeth

Returns the buffer element of the Teeths line by the specified index.

```
double  Teeth(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the Teeths line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Lips

Returns the buffer element of the Lips line by the specified index.

```
double  Lips(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the Lips line of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_ALLIGATOR for CiAlligator).

CiAO

CiAO is a class intended for using the Awesome Oscillator technical indicator.

Description

The CiAO class provides the creation, setup and access to the data of the Awesome Oscillator indicator.

Declaration

```
class CiAO: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Class Methods

Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,      // Symbol
   ENUM_TIMEFRAMES  period       // Period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_AO for CiAO).

CiFractals

CiFractals is a class intended for using the Fractals technical indicator.

Description

The CiFractals class provides the creation, setup and access to the data of the Fractals indicator.

Declaration

```
class CiFractals: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Class Methods

Create Methods	
Create	Creates the indicator
Data Access Methods	
Upper	Returns the buffer element of the upper buffer
Lower	Returns the buffer element of the lower buffer
Input/output	
virtual Type	Returns the object type identifier

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string           symbol,     // Symbol
   ENUM_TIMEFRAMES  period      // Period
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Upper

Returns the buffer element of the upper buffer by the specified index.

```
double  Upper(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the upper buffer of the specified index, or EMPTY_VALUE if there isn't any correct data.

Lower

Returns the buffer element of the lower buffer by the specified index.

```
double  Lower(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the lower buffer of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_FRACTALS for CiFractals).

CiGator

CiGator is a class intended for using the Gator Oscillator technical indicator.

Description

The CiGator class provides the creation, setup and access to the data of the Gator Oscillator indicator.

Declaration

```
class CiGator: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Class Methods

Attributes	
JawPeriod	Returns the averaging period for the Jaws line
JawShift	Returns the horizontal shift of the Jaws line
TeethPeriod	Returns the averaging period for the Teeths line
TeethShift	Returns the horizontal shift of the Teeths line
LipsPeriod	Returns the averaging period for the Lips line
LipsShift	Returns the horizontal shift of the Lips line
MaMethod	Returns the averaging method
Applied	Returns the price type or handle to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Upper	Returns the buffer element of the upper buffer
Lower	Returns the buffer element of the lower buffer
Input/output	
virtual Type	Returns the object type identifier

JawPeriod

Returns the averaging period for the Jaws line.

```
int  JawPeriod() const
```

Returned value

Returns the averaging period for the Jaws line, defined at the indicator creation.

JawShift

Returns the horizontal shift of the Jaws line.

```
int  JawShift() const
```

Returned value

Horizontal shift of the Jaws line, defined at the indicator creation.

TeethPeriod

Returns the averaging period for the Teeth line.

```
int  TeethPeriod() const
```

Returned value

Returns the averaging period for the Teeth line, defined at the indicator creation.

TeethShift

Returns the horizontal shift of the Teeths line.

```
int  TeethShift() const
```

Returned value

Horizontal shift of the Teeths line, defined at the indicator creation.

LipsPeriod

Returns the averaging period for the Lips line.

```
int  LipsPeriod() const
```

Returned value

Returns the averaging period for the Lips line, defined at the indicator creation.

LipsShift

Returns the horizontal shift of the Lips line.

```
int  LipsShift() const
```

Returned value

Horizontal shift of the Lips line, defined at the indicator creation.

MaMethod

Returns the averaging method.

```
ENUM_MA_METHOD  MaMethod() const
```

Returned value

Returns the averaging method, defined at the indicator creation.

Applied

Returns the price type or handle to apply.

```
int  Applied() const
```

Returned value

Price type or handle to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string            symbol,           // Symbol
   ENUM_TIMEFRAMES   period,           // Period
   int               jaw_period,       // Jaws period
   int               jaw_shift,        // Jaws shift
   int               teeth_period,     // Teeths period
   int               teeth_shift,      // Teeths shift
   int               lips_period,      // Lips period
   int               lips_shift,       // Lips shift
   ENUM_MA_METHOD    ma_method,        // Averaging method
   int               applied           // Volume type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

jaw_period

[in] Jaws period.

jaw_shift

[in] Jaws shift.

teeth_period

[in] Teeths period.

teeth_shift

[in] Teeths shift.

lips_period

[in] Lips period.

lips_shift

[in] Lips shift.

ma_method

[in] Averaging method (ENUM_MA_METHOD enumeration).

applied

[in] Volume type to apply.

Returned value

true if successful, false if indicator hasn't been created.

Upper

Returns the buffer element of the upper buffer by the specified index.

```
double  Upper(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the upper buffer of the specified index, or EMPTY_VALUE if there isn't any correct data.

Lower

Returns the buffer element of the lower buffer by the specified index.

```
double  Upper(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

The buffer element of the lower buffer of the specified index, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_GATOR for CiGator).

CiBWMFI

CiBWMFI is a class intended for using the Market Facilitation Index by Bill Williams technical indicator.

Description

The CiBWMFI class provides the creation, setup and access to the data of the Market Facilitation Index by Bill Williams indicator.

Declaration

```
class CiBWMFI: public CIndicator
```

Title

```
#include <Indicators\BillWilliams.mqh>
```

Class Methods

Attributes	
Applied	Returns the volume type to apply
Create Methods	
Create	Creates the indicator
Data Access Methods	
Main	Returns the buffer element
Input/output	
virtual Type	Returns the object type identifier

Applied

Returns the volume type to apply.

```
ENUM_APPLIED_VOLUME  Applied() const
```

Returned value

Volume type to apply, defined at the indicator creation.

Create

It creates the indicator with specified parameters.

```
bool  Create(
   string                symbol,      // Symbol
   ENUM_TIMEFRAMES       period,      // Period
   ENUM_APPLIED_VOLUME   applied      // Volume type to apply
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

applied

[in] Volume type to apply (ENUM_APPLIED_VOLUME enumeration).

Returned value

true if successful, false if indicator hasn't been created.

Main

Returns the buffer element by the specified index.

```
double  Main(
   int  index      // Index
   )
```

Parameters

index

[in] Element index.

Returned value

Buffer element of the specified index if successful, or EMPTY_VALUE if there isn't any correct data.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_BWMFI for CiBWMFI).

CiCustom

CiCustom is a class intended for using the custom technical indicators.

Description

The CiCustom class provides the creation, setup and access to the data of the custom indicator.

Declaration

```
class CiCustom: public CIndicator
```

Title

```
#include <Indicators\Custom.mqh>
```

Class Methods

Attributes	
NumBuffers	Sets the number of buffers
NumParams	Gets the number of parameters
ParamType	Gets the type of the specified parameter
ParamLong	Gets the value of the specified parameter of integer type
ParamDouble	Gets the value of the specified parameter of double type
ParamString	Gets the value of the specified parameter of string type
Input/output	
virtual Type	Gets the object type identifier

NumBuffers

Sets the number of buffers.

```
bool  NumBuffers(
   int  buffers      // number of buffers
   )
```

Returned value

true if successful, false if buffers haven't been set.

NumParams

Gets the number of parameters.

```
int  NumParams() const
```

Returned value

Number of parameters, used in creation of the indicator.

ParamType

Gets a type of the parameter with specified index.

```
ENUM_DATATYPE  ParamType(
   int  index      // parameter index
   ) const
```

Parameters

index

[in] Parameter index.

Returned value

Returns the data type (value of ENUM_DATATYPE enumeration) of the parameter with specified index, used in indicator creation.

Note

If parameter index is invalid, it returns WRONG_VALUE.

ParamLong

Gets the value of specified parameter of long type.

```
long  ParamLong(
   int  index      // index
   ) const
```

Parameters

index

[in] Parameter index.

Returned value

The value of specified parameter of long type, used in creation of the indicator.

Note

If the number is invalid, it returns 0.

ParamDouble

Gets the value of specified parameter of double type.

```
double  ParamDouble(
   int  index      // index
   ) const
```

Parameters

index

[in] Parameter index.

Returned value

The value of specified parameter of double type, used in creation of the indicator.

Note

If the number is invalid, it returns EMPTY_VALUE.

ParamString

Gets the value of specified parameter of string type.

```
string  ParamString(
   int  index      // index
   ) const
```

Parameters

index

[in] Parameter index.

Returned value

The value of specified string parameter, used in creation of the indicator.

Note

If the number is invalid, it returns an empty string.

Type

Returns the object type identifier.

```
virtual int  Type() const
```

Returned value

Object type identifier (IND_CUSTOM for CiCustom).

Trade Classes

This section contains technical details of working with trade classes and description of the relevant components of the MQL5 standard library.

Using trade classes will save time when creating custom programs (Expert Advisors).

MQL5 Standard Library (in terms of data sets) is placed in the terminal working directory, in the Include\Arrays folder.

Class/Group	Description
CAccountInfo	Class for working with trade account properties
CSymbolInfo	Class for working with trade instrument properties
COrderInfo	Class for working with pending order properties
CHistoryOrderInfo	Class for working with history order properties
CPositionInfo	Class for working with open position properties
CDealInfo	Class for working with history deal properties
CTrade	Class for trade operations execution
CTerminalInfo	Class for getting the properties of mql5 program environment

CAccountInfo

CAccountInfo is a class for easy access to the currently opened trade account properties.

Description

CAccountInfo class provides easy access to the currently opened trade account properties.

Declaration

```
class CAccountInfo : public CObject
```

Title

```
#include <Trade\AccountInfo.mqh>
```

Class methods by groups

Access to integer type properties	
Login	Gets the account number
TradeMode	Gets the trade mode
TradeModeDescription	Gets the trade mode as a string
Leverage	Gets the amount of given leverage
MarginMode	Gets the mode of account stop out
MarginModeDescription	Gets the description of account stop out mode
TradeAllowed	Gets the flag of trade allowance
TradeExpert	Gets the flag of automated trade allowance
LimitOrders	Gets the maximal number of allowed pending orders
Access to double type properties	
Balance	Gets the balance of account
Credit	Gets the amount of given credit
Profit	Gets the amount of current profit on account
Equity	Gets the amount of current equity on account
Margin	Gets the amount of reserved margin
FreeMargin	Gets the amount of free margin
MarginLevel	Gets the level of margin
MarginCall	Gets the level of margin for deposit
MarginStopOut	Gets the level of margin for Stop Out

Access to text properties	
Name	Gets the client name
Server	Gets the trade server name
Currency	Gets the deposit currency name
Company	Gets the company name, that serves an account
Access to MQL5 API functions	
Integer	Gets the value of specified integer type property
Double	Gets the value of specified double type property
String	Gets value of specified string type property
Additional methods	
OrderProfitCheck	Gets the evaluated profit, based on the parameters passed
MarginCheck	Gets the amount of margin, required to execute trade operation
FreeMarginCheck	Gets the amount of free margin, left after execution of trade operation
MaxLotCheck	Gets the maximal possible volume of trade operation

Login

Gets the account number.

```
long  Login() const
```

Returned value

Account number.

TradeMode

Gets the trade mode.

```
ENUM_ACCOUNT_TRADE_MODE  TradeMode() const
```

Returned value

Trade mode (value of ENUM_ACCOUNT_TRADE_MODE enumeration).

TradeModeDescription

Gets the trade mode as a string.

```
string  TradeModeDescription() const
```

Returned value

Trade mode as a string.

Leverage

Gets the amount of given leverage.

```
long  Leverage() const
```

Returned value

Amount of given leverage.

MarginMode

Gets the mode of account Stop Out.

```
ENUM_ACCOUNT_STOPOUT_MODE  MarginMode() const
```

Returned value

Account Stop Out mode (value of ENUM_ACCOUNT_STOPOUT_MODE enumeration).

MarginModeDescription

Gets the mode of setting minimal margin level as a string.

```
string  MarginModeDescription() const
```

Returned value

Mode of setting minimal margin level as a string.

TradeAllowed

Gets the flag of trade allowance.

```
bool  TradeAllowed() const
```

Returned value

Flag of trade allowance.

TradeExpert

Gets the flag of automated trade allowance.

```
bool  TradeExpert() const
```

Returned value

Flag of automated trade allowance.

LimitOrders

Gets the maximal number of allowed pending orders

```
int  LimitOrders() const
```

Returned value

The maximal number of allowed pending orders.

Note

0 - no limits.

Balance

Gets the balance of account.

```
double  Balance() const
```

Returned value

The balance of account (in deposit currency).

Credit

Gets the amount of given credit.

```
double  Credit() const
```

Returned value

Amount of given credit (in deposit currency).

Profit

Gets the amount of current profit on account.

```
double  Profit() const
```

Returned value

Amount of current profit on account (in deposit currency).

Equity

Gets the amount of current equity on account.

```
double  Equity() const
```

Returned value

Amount of current equity on account (in deposit currency).

Margin

Gets the amount of reserved margin.

```
double  Margin() const
```

Returned value

Amount of reserved margin (in deposit currency).

FreeMargin

Gets the amount of free margin.

```
double  FreeMargin() const
```

Returned value

Amount of free margin (in deposit currency).

MarginLevel

Gets the level of margin.

```
double  MarginLevel() const
```

Returned value

Level of margin.

MarginCall

Gets the level of margin for a deposit.

```
double  MarginCall() const
```

Returned value

Level of margin for a deposit.

MarginStopOut

Gets the level of margin for Stop Out.

```
double  MarginStopOut() const
```

Returned value

Level of margin for Stop Out.

Name

Gets the client name.

```
string  Name() const
```

Returned value

Client name.

Server

Gets the trade server name.

```
string  Server() const
```

Returned value

Trade server name.

Currency

Gets the deposit currency name.

```
string  Currency() const
```

Returned value

Deposit currency name.

Company

Gets the company name, that serves an account.

```
string  Company() const
```

Returned value

Company name, that serves an account.

InfoInteger

Gets the value of specified integer type property.

```
long  InfoInteger(
   ENUM_ACCOUNT_INFO_INTEGER  prop_id     // property ID
   ) const
```

Parameters

prop_id

[in] Identifier of the property. The value can be one of the values of ENUM_ACCOUNT_INFO_INTEGER enumeration.

Returned value

Value of long type.

InfoDouble

Gets the value of specified double type property.

```
double  InfoDouble(
   ENUM_ACCOUNT_INFO_DOUBLE  prop_id     // property ID
   ) const
```

Parameters

prop_id

[in] Identifier of the property. The value can be one of the values of ENUM_ACCOUNT_INFO_DOUBLE enumeration.

Returned value

Value of double type.

InfoString

Gets the value of specified string type property.

```
string  InfoString(
   ENUM_ACCOUNT_INFO_STRING  prop_id     // property ID
   ) const
```

Parameters

prop_id

[in] Identifier of the property. The value can be one of the values of ENUM_ACCOUNT_INFO_STRING enumeration.

Returned value

Value of string type.

OrderProfitCheck

The function calculates the profit for the current account, based on the parameters passed. The function is used for pre-evaluation of the result of a trade operation. The value is returned in the account currency.

```
double  OrderProfitCheck(
   const string        symbol,              // symbol
   ENUM_ORDER_TYPE     trade_operation,     // operation type (ORDER_TYPE_BUY or ORDE
   double              volume,              // volume
   double              price_open,          // open price
   double              price_close          // close price
   ) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation (ENUM_ORDER_TYPE enumeration).

volume

[in] Volume of trade operation.

price_open

[in] Open price.

price_close

[in] Close price.

Returned value

If successful, it returns amount of profit or EMPTY_VALUE in the case of error.

MarginCheck

Gets the amount of margin, required for trade operation.

```
double  MarginCheck(
   const string         symbol,              // symbol
   ENUM_ORDER_TYPE      trade_operation,     // operation
   double               volume,              // volume
   double               price                // price
   ) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation (ENUM_ORDER_TYPE enumeration).

volume

[in] Volume of trade operation.

price

[in] Price of trade operation.

Returned value

Amount of margin, required for trade operation.

FreeMarginCheck

Gets the amount of free margin, left after trade operation.

```
double  FreeMarginCheck(
   const string         symbol,              // symbol
   ENUM_ORDER_TYPE      trade_operation,     // operation
   double               volume,              // volume
   double               price                // price
   ) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation (ENUM_ORDER_TYPE enumeration).

volume

[in] Volume of trade operation.

price

[in] Price of trade operation.

Returned value

Amount of free margin, left after trade operation.

MaxLotCheck

Gets the maximum possible volume of trade operation.

```
double  MaxLotCheck(
   const string         symbol,              // symbol
   ENUM_ORDER_TYPE      trade_operation,     // operation
   double               price,               // price
   double               percent=100          // percent of available margin (0-100%),
   ) const
```

Parameters

symbol

[in] Symbol for trade operation.

trade_operation

[in] Type of trade operation (ENUM_ORDER_TYPE enumeration).

price

[in] Price of trade operation.

percent=100

[in] Percent of available margin (0-100%), used for trade operation.

Returned value

Maximum possible volume of trade operation.

CSymbolInfo

CSymbolInfo is a class for easy access to the symbol properties.

Description

CSymbolInfo class provides access to the symbol properties.

Declaration

```
class CSymbolInfo : public CObject
```

Title

```
#include <Trade\SymbolInfo.mqh>
```

Class methods by groups

Controlling	
Refresh	Refreshes the symbol data
RefreshRates	Refreshes the symbol quotes
Properties	
Name	Gets/sets symbol name
Select	Gets/sets the "Market Watch" symbol flag
IsSynchronized	Checks the symbol synchronization with server
Volumes	
Volume	Gets the volume of last deal
VolumeHigh	Gets the maximal volume for a day
VolumeLow	Gets the minimal volume for a day
Miscellaneous	
Time	Gets the time of last quote
Spread	Gets the amount of spread (in points)
SpreadFloat	Gets the flag of floating spread
TickBookDepth	Gets the depth of ticks saving
Levels	
StopsLevel	Gets the minimal indent for orders (in points)
FreezeLevel	Gets the distance of freezing trade operations (in points)
Bid prices	

Bid	Gets the current Bid price
BidHigh	Gets the maximal Bid price for a day
BidLow	Gets the minimal Bid price for a day
Ask prices	
Ask	Gets the current Ask price
AskHigh	Gets the maximal Ask price for a day
AskLow	Gets the minimal Ask price for a day
Prices	
Last	Gets the current Last price
LastHigh	Gets the maximal Last price for a day
LastLow	Gets the minimal Last price for a day
Trade modes	
TradeCalcMode	Gets the mode of contract cost calculation
TradeCalcModeDescription	Gets the mode of contract cost calculation as a string
TradeMode	Gets the type of order execution
TradeModeDescription	Gets the type of order execution as a string
TradeExecution	Gets the closing of deals mode
TradeExecutionDescription	Gets the closing of deals mode as a string
Swaps	
SwapMode	Gets the swap calculation model
SwapModeDescription	Gets the swap calculation model as a string
SwapRollover3days	Gets the day of triple swap charge
SwapRollover3daysDescription	Gets the day of triple swap charge as a string
Margins and flags	
MarginInitial	Gets the value of initial margin
MarginMaintenance	Gets the value of maintenance margin
MarginLong	Gets the rate of margin charging for long positions
MarginShort	Gets the rate of margin charging for short positions
MarginLimit	Gets the rate of margin charging for Limit orders
MarginStop	Gets the rate of margin charging for Stop

	orders
MarginStopLimit	Gets the rate of margin charging for StopLimit orders
TradeTimeFlags	Gets the flags of the order expiration allowed modes
TradeFillFlags	Gets the flags of the order filling allowed modes
Quantization	
Digits	Gets the number of digits after period
Point	Gets the value of one point
TickValue	Gets the cost of tick (minimal change of price)
TickValueProfit	Gets the calculated tick price for a profitable position
TickValueLoss	Gets the calculated tick price for a losing position
TickSize	Gets the minimal change of price
Contracts sizes	
ContractSize	Gets the amount of trade contract
LotsMin	Gets the minimal volume to close a deal
LotsMax	Gets the maximal volume to close a deal
LotsStep	Gets the minimal step of volume change to close a deal
LotsLimit	Gets the maximal allowed volume of opened position and pending orders (direction insensitive) for one symbol
Swaps sizes	
SwapLong	Gets the value of long position swap
SwapShort	Gets the value of short position swap
Text properties	
CurrencyBase	Gets the name of symbol base currency
CurrencyProfit	Gets the profit currency name
CurrencyMargin	Gets the margin currency name
Bank	Gets the name of current quote source
Description	Gets the string description of symbol
Path	Gets the path in symbols tree
Symbol properties	

SessionDeals	Gets the number of deals in the current session
SessionBuyOrders	Gets the number of Buy orders at the moment
SessionSellOrders	Gets the number of Sell orders at the moment
SessionTurnover	Gets the summary turnover of the current session
SessionInterest	Gets the summary open interest of the current session
SessionBuyOrdersVolume	Gets the current volume of Buy orders
SessionSellOrdersVolume	Gets the current volume of Sell orders
SessionOpen	Gets the open price of the current session
SessionClose	Gets the close price of the current session
SessionAW	Gets the average weighted price of the current session
SessionPriceSettlement	Gets the settlement price of the current session
SessionPriceLimitMin	Gets the minimal price of the current session
SessionPriceLimitMax	Gets the maximal price of the current session
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets the value of specified string type property
Service functions	
NormalizePrice	Returns the value of price, normalized using the symbol properties

Refresh

Refreshes the symbol data.

```
void  Refresh()
```

Returned value

None.

Note

The symbol should be selected by Name method.

RefreshRates

Refreshes the symbol quotes data.

```
bool  RefreshRates()
```

Returned value

true - in case of success, false - if unable to refresh quotes.

Note

The symbol should be selected by Name method.

Name

Gets symbol name.

```
string  Name() const
```

Returned value

Symbol name.

Name

Sets symbol name.

```
void  Name(string name)
```

Returned value

None.

Select

Gets the "Market Watch" symbol flag.

```
bool  Select() const
```

Returned value

Gets the "Market Watch" symbol flag.

Select

Sets the "Market Watch" symbol flag.

```
bool  Select()
```

Returned value

true - in case of success, false - if unable to change flag.

IsSynchronized

Checks the symbol synchronization with server.

```
bool  IsSynchronized() const
```

Returned value

true - if the symbol is synchronized with server, false - if not.

Note

The symbol should be selected by Name method.

Volume

Gets the volume of last deal.

```
long  Volume() const
```

Returned value

Volume of last deal.

Note

The symbol should be selected by Name method.

VolumeHigh

Gets the maximal volume of the day.

```
long  VolumeHigh() const
```

Returned value

Maximal volume of the day.

Note

The symbol should be selected by Name method.

VolumeLow

Gets the minimal volume of the day.

```
long  VolumeLow() const
```

Returned value

Minimal volume of the day.

Note

The symbol should be selected by Name method.

Time

Gets the time of last quote.

```
datetime  Time() const
```

Returned value

Time of last quote.

Note

The symbol should be selected by Name method.

Spread

Gets the amount of spread (in points).

```
int  Spread() const
```

Returned value

Gets the amount of spread (in points).

Note

The symbol should be selected by Name method.

SpreadFloat

Gets the flag of floating spread.

```
bool  SpreadFloat() const
```

Returned value

Flag of floating spread.

Note

The symbol should be selected by Name method.

TicksBookDepth

Gets the depth of ticks saving.

```
int  TicksBookDepth() const
```

Returned value

Depth of ticks saving.

Note

The symbol should be selected by Name method.

StopsLevel

Gets the minimal stop level for orders (in points).

```
int  StopsLevel() const
```

Returned value

Minimal stop level for orders (in points).

Note

The symbol should be selected by Name method.

FreezeLevel

Gets the freeze level (in points).

```
int  FreezeLevel() const
```

Returned value

Distance of freeze level (in points).

Note

The symbol should be selected by Name method.

Bid

Gets the current Bid price.

```
double  Bid() const
```

Returned value

Current Bid price.

Note

The symbol should be selected by Name method.

BidHigh

Gets the maximal Bid price of the day.

```
double  BidHigh() const
```

Returned value

Maximal Bid price of the day.

Note

The symbol should be selected by Name method.

BidLow

Gets the minimal Bid price of the day.

```
double  BidLow() const
```

Returned value

Minimal Bid price of the day.

Note

The symbol should be selected by Name method.

Ask

Gets the current Ask price.

```
double  Ask() const
```

Returned value

Current Ask price.

Note

The symbol should be selected by Name method.

AskHigh

Gets the maximal Ask price for a day.

```
double  AskHigh() const
```

Returned value

Maximal Ask price of the day.

Note

The symbol should be selected by Name method.

AskLow

Gets the minimal Ask price for a day.

```
double  AskLow() const
```

Returned value

Minimal Ask price of the day.

Note

The symbol should be selected by Name method.

Last

Gets the current Last price.

```
double  Last() const
```

Returned value

Current Last price.

LastHigh

Gets the maximal Last price of the day.

```
double  LastHigh() const
```

Returned value

Maximal Last price of the day.

Note

The symbol should be selected by Name method.

LastLow

Gets the minimal Last price of the day.

```
double  LastLow() const
```

Returned value

Minimal Last price of the day.

Note

The symbol should be selected by Name method.

TradeCalcMode

Gets the mode of contract cost calculation.

```
ENUM_SYMBOL_CALC_MODE  TradeCalcMode() const
```

Returned value

Mode of contract cost calculation (value of ENUM_SYMBOL_CALC_MODE enumeration).

Note

The symbol should be selected by Name method.

TradeCalcModeDescription

Gets the mode of contract cost calculation as a string.

```
string  TradeCalcModeDescription() const
```

Returned value

Mode of contract cost calculation as a string.

Note

The symbol should be selected by Name method.

TradeMode

Gets the order execution type.

```
ENUM_SYMBOL_TRADE_MODE  TradeMode() const
```

Returned value

Order execution type (value of ENUM_SYMBOL_TRADE_MODE enumeration).

Note

The symbol should be selected by Name method.

TradeModeDescription

Gets the trade mode as a string.

```
string  TradeModeDescription() const
```

Returned value

Trade mode as a string.

Note

The symbol should be selected by Name method.

TradeExecution

Gets the trade execution mode.

```
ENUM_SYMBOL_TRADE_EXECUTION  TradeExecution() const
```

Returned value

Trade execution mode (value of ENUM_SYMBOL_TRADE_EXECUTION enumeration).

Note

The symbol should be selected by Name method.

TradeExecutionDescription

Gets the description of trade execution mode as a string.

```
string  TradeExecutionDescription() const
```

Returned value

Trade execution mode as a string.

Note

The symbol should be selected by Name method.

SwapMode

Gets the swap calculation mode.

```
ENUM_SYMBOL_SWAP_MODE  SwapMode() const
```

Returned value

Swap calculation mode (value of ENUM_SYMBOL_SWAP_MODE enumeration).

Note

The symbol should be selected by Name method.

SwapModeDescription

Gets the swap mode description as a string.

```
string  SwapModeDescription() const
```

Returned value

Swap mode descriprion as a string.

Note

The symbol should be selected by Name method.

SwapRollover3days

Gets the swap rollover day.

```
ENUM_DAY_OF_WEEK  SwapRollover3days() const
```

Returned value

Swap rollover day (value of ENUM_DAY_OF_WEEK enumeration).

Note

The symbol should be selected by Name method.

SwapRollover3daysDescription

Gets the swap rollover day as a string.

```
string  SwapRollover3daysDescription() const
```

Returned value

Swap rollover day as a string.

Note

The symbol should be selected by Name method.

MarginInitial

Gets the value of initial margin.

```
double  MarginInitial()
```

Returned value

Value of initial margin.

Note

It points the amount of margin (in margin currency of instrument), that is charged from one lot. Used to check client's equity, when he enters the market.

The symbol should be selected by Name method.

MarginMaintenance

Gets the value of maintenance margin.

```
double  MarginMaintenance()
```

Returned value

Value of maintenance margin.

Note

It points the amount of margin (in margin currency of instrument), that is charged from one lot. Used to check client's equity, when the account state is changed. If the maintenance margin is equal to 0, then the initial margin is used.

The symbol should be selected by Name method.

MarginLong

Gets the rate of margin charging on long positons.

```
double  MarginLong() const
```

Returned value

Rate of margin charging on long positons.

Note

The symbol should be selected by Name method.

MarginShort

Gets the rate of margin charging on short positons.

```
double  MarginShort() const
```

Returned value

Rate of margin charging on short positons.

Note

The symbol should be selected by Name method.

MarginLimit

Gets the rate of margin charging on Limit orders.

```
double  MarginLimit() const
```

Returned value

Rate of margin charging on Limit orders.

Note

The symbol should be selected by Name method.

MarginStop

Gets the rate of margin charging on Stop orders.

```
double  MarginStop() const
```

Returned value

Rate of margin charging on Stop orders.

Note

The symbol should be selected by Name method.

MarginStopLimit

Gets the rate of margin charging on Stop Limit orders.

```
double  MarginStopLimit() const
```

Returned value

Rate of margin charging on Stop Limit orders.

Note

The symbol should be selected by Name method.

TradeTimeFlags

Gets the flags of the order expiration allowed modes.

```
int  TradeTimeFlags() const
```

Returned value

Flags of the order expiration allowed modes.

Note

The symbol should be selected by Name method.

TradeFillFlags

Gets the flags of the order filling allowed modes.

```
int  TradeFillFlags() const
```

Returned value

Flags of the order filling allowed modes.

Note

The symbol should be selected by Name method.

Digits

Gets the number of digits after period.

```
int  Digits() const
```

Returned value

Gets the number of digits after period.

Note

The symbol should be selected by Name method.

Point

Gets the value of one point.

```
double  Point() const
```

Returned value

Value of one point.

Note

The symbol should be selected by Name method.

TickValue

Gets the cost of tick (minimal change of price).

```
double  TickValue() const
```

Returned value

Cost of tick (minimal change of price).

Note

The symbol should be selected by Name method.

TickValueProfit

Gets the calculated tick price for a profitable position.

```
double  TickValueProfit() const
```

Returned value

The calculated tick price for a profitable position.

Note

The symbol should be selected by Name method.

TickValueLoss

Gets the calculated tick price for a losing position.

```
double  TickValueLoss() const
```

Returned value

The calculated tick price for a losing position.

Note

The symbol should be selected by Name method.

TickSize

Gets the minimal change of price.

```
double  TickSize() const
```

Returned value

Minimal change of price.

Note

The symbol should be selected by Name method.

ContractSize

Gets the amount of trade contract.

```
double  ContractSize() const
```

Returned value

Amount of trade contract.

Note

The symbol should be selected by Name method.

LotsMin

Gets the minimal volume to close a deal.

```
double  LotsMin() const
```

Returned value

Minimal volume to close a deal.

Note

The symbol should be selected by Name method.

LotsMax

Gets the maximal volume to close a deal.

```
double  LotsMax() const
```

Returned value

Maximal volume to close a deal.

Note

The symbol should be selected by Name method.

LotsStep

Gets the minimal step of volume change to close a deal.

```
double  LotsStep() const
```

Returned value

Minimal step of volume change to close a deal.

Note

The symbol should be selected by Name method.

LotsLimit

Gets the maximal allowed volume of opened position and pending orders (direction insensitive) for one symbol.

```
double  LotsLimit() const
```

Returned value

The maximal allowed volume of opened position and pending orders (direction insensitive) for one symbol.

Note

The symbol should be selected by Name method.

SwapLong

Gets the value of long position swap.

```
double  SwapLong() const
```

Returned value

Value of long position swap.

Note

The symbol should be selected by Name method.

SwapShort

Gets the value of short position swap.

```
double  SwapShort() const
```

Returned value

Value of short position swap.

Note

The symbol should be selected by Name method.

CurrencyBase

Gets the name of symbol base currency.

```
string  CurrencyBase() const
```

Returned value

Name of symbol base currency.

Note

The symbol should be selected by Name method.

CurrencyProfit

Gets the profit currency name.

```
string  CurrencyProfit() const
```

Returned value

Profit currency name.

Note

The symbol should be selected by Name method.

CurrencyMargin

Gets the margin currency name.

```
string  CurrencyMargin() const
```

Returned value

Margin currency name.

Note

The symbol should be selected by Name method.

Bank

Gets the name of current quote source.

```
string  Bank() const
```

Returned value

Name of current quote source.

Note

The symbol should be selected by Name method.

Description

Gets the string description of symbol.

```
string  Description() const
```

Returned value

String description of symbol.

Note

The symbol should be selected by Name method.

Path

Gets the path in symbols tree.

```
string  Path() const
```

Returned value

Gets the path in symbols tree.

Note

The symbol should be selected by Name method.

SessionDeals

Gets the number of deals in the current session.

```
long  SessionDeals() const
```

Returned value

Number of deals in the current session.

Note

The symbol should be selected by Name method.

SessionBuyOrders

Gets the number of Buy orders at the moment.

```
long  SessionBuyOrders() const
```

Returned value

Number of Buy orders at the moment.

Note

The symbol should be selected by Name method.

SessionSellOrders

Gets then number of Sell orders at the moment.

```
long  SessionSellOrders() const
```

Returned value

Number of Sell orders at the moment.

Note

The symbol should be selected by Name method.

SessionTurnover

Gets summary turnover of the current session.

```
double  SessionTurnover() const
```

Returned value

Summary turnover of the current session.

Note

The symbol should be selected by Name method.

SessionInterest

Gets the summary open interest of the current session.

```
double  SessionInterest() const
```

Returned value

Summary open interest of the current session.

Note

The symbol should be selected by Name method.

SessionBuyOrdersVolume

Gets the current volume of Buy orders.

```
double  SessionBuyOrdersVolume() const
```

Returned value

Current volume of Buy orders.

Note

The symbol should be selected by Name method.

SessionSellOrdersVolume

Gets the current volume of Sell orders.

```
double  SessionSellOrdersVolume() const
```

Returned value

Current volume of Sell orders.

Note

The symbol should be selected by Name method.

SessionOpen

Gets the open price of the current session.

```
double  SessionOpen() const
```

Returned value

Open price of the current session.

Note

The symbol should be selected by Name method.

SessionClose

Gets the close price of the current session.

```
double  SessionClose() const
```

Returned value

Close price of the current session.

Note

The symbol should be selected by Name method.

SessionAW

Gets the average weighted price of the current session.

```
double  SessionAW() const
```

Returned value

Average weighted price of the current session.

Note

The symbol should be selected by Name method.

SessionPriceSettlement

Gets the settlement price of the current session.

```
double  SessionPriceSettlement() const
```

Returned value

Settlement price of the current session.

Note

The symbol should be selected by Name method.

SessionPriceLimitMin

Gets the minimal price of the current session.

```
double  SessionPriceLimitMin() const
```

Returned value

Minimal price of the current session.

Note

The symbol should be selected by Name method.

SessionPriceLimitMax

Gets the maximal price of the current session.

```
double  SessionPriceLimitMax() const
```

Returned value

Maximal price of the current session.

Note

The symbol should be selected by Name method.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_SYMBOL_INFO_INTEGER  prop_id,      // property ID
   long&                     var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of integer type property (value of ENUM_SYMBOL_INFO_INTEGER enumeration).

var

[out] Reference to long type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The symbol should be selected by Name method.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_SYMBOL_INFO_DOUBLE  prop_id,      // property ID
   double&                  var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property (value of ENUM_SYMBOL_INFO_DOUBLE enumeration).

var

[out] Reference to double type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The symbol should be selected by Name method.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_SYMBOL_INFO_STRING  prop_id,      // property ID
   string&                  var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of text property.

var

[out] Reference to string type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The symbol should be selected by Name method.

NormalizePrice

Returns the value of price, normalized using the symbol properties.

```
double  NormalizePrice(
   double  price   // price
   ) const
```

Parameters

price

[in] Price.

Returned value

Normalized price.

Note

The symbol should be selected by Name method.

COrderInfo

COrderInfo is a class for easy access to the pending order properties.

Description

COrderInfo class provides access to the pending order properties.

Declaration

```
class COrderInfo : public CObject
```

Title

```
#include <Trade\OrderInfo.mqh>
```

Class methods by groups

Access to integer type properties	
Ticket	Gets the ticket of an order, previously selected for access
TimeSetup	Gets the time of order placement
TimeSetupMsc	Получает время установки ордера в миллисекундах с 01.01.1970
OrderType	Gets the order type
OrderTypeDescription	Gets the order type as a string
State	Gets the order state
StateDescription	Gets the order state as a string
TimeExpiration	Gets the time of order expiration
TimeDone	Gets the time of order execution or cancellation
TimeDoneMsc	Получает время исполнения или снятия ордера в миллисекундах с 01.01.1970
TypeFilling	Gets the type of order execution by remainder
TypeFillingDescription	Gets the type of order execution by remainder as a string
TypeTime	Gets the type of order at the time of the expiration
TypeTimeDescription	Gets the order type by expiration time as a string
Magic	Gets the ID of expert, that placed the order
PositionId	Gets the ID of position

Access to double type properties	
VolumeInitial	Gets the initial volume of order
VolumeCurrent	Gets the unfilled volume of order
PriceOpen	Gets the order price
StopLoss	Gets the order's Stop Loss
TakeProfit	Gets the order's Take Profit
PriceCurrent	Gets the current price by order symbol
PriceStopLimit	Gets the price of setting limit order
Access to text properties	
Symbol	Gets the name of order symbol
Comment	Gets the order comment
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets value of specified string type property
State	
StoreState	Saves the order parameters
CheckState	Checks the current parameters against the saved parameters
Selection	
Select	Selects an order by ticket for further access to its properties
SelectByIndex	Selects an order by index for further access to its properties

Ticket

Gets the ticket of an order, previously selected for access using the Select method.

```
ulong  Ticket() const
```

Returned value

Order ticket if successful, otherwise - ULONG_MAX.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeSetup

Gets the time of order placement.

```
datetime  TimeSetup() const
```

Returned value

Time of order placement.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeSetupMsc

Получает время установки ордера на исполнение в миллисекундах с 01.01.1970.

```
ulong  TimeSetupMsc() const
```

Возвращаемое значение

Время установки ордера на исполнение в миллисекундах с 01.01.1970.

Примечание

Ордер должен быть предварительно выбран для доступа методом Select (по тикету) или SelectByIndex (по индексу).

OrderType

Gets the order type.

```
ENUM_ORDER_TYPE  OrderType()
```

Returned value

Order type (value of ENUM_ORDER_TYPE enumeration).

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeDescription

Gets the order type as a string.

```
string  TypeDescription() const
```

Returned value

Order type as a string.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

State

Gets the order state.

```
ENUM_ORDER_STATE  State() const
```

Returned value

Order state (value of ENUM_ORDER_STATE enumeration).

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

StateDescription

Gets the order state as a string.

```
string  StateDescription() const
```

Returned value

Order state as a string.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeExpiration

Gets the order expiration time.

```
datetime  TimeExpiration() const
```

Returned value

Order expiration time, set on its placement.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeDone

Gets the time of order execution or cancellation.

```
datetime  TimeDone() const
```

Returned value

Time of order execution or cancellation.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeDoneMsc

Получает время исполнения или снятия ордера в миллисекундах с 01.01.1970.

```
ulong  TimeDoneMsc() const
```

Возвращаемое значение

Время исполнения или снятия ордера в миллисекундах с 01.01.1970.

Примечание

Ордер должен быть предварительно выбран для доступа методом Select (по тикету) или SelectByIndex (по индексу).

TypeFilling

Gets the order filling type.

```
ENUM_ORDER_TYPE_FILLING  TypeFilling() const
```

Returned value

Order filling type (value of ENUM_ORDER_TYPE_FILLING enumeration).

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeFillingDescription

Gets the order filling type as a string.

```
string  TypeFillingDescription() const
```

Returned value

Order filling type as a string.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeTime

Gets the type of order at the time of the expiration.

```
ENUM_ORDER_TYPE_TIME  TypeTime() const
```

Returned value

Type of order at the time of the expiration.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeTimeDescription

Gets the order type by expiration time as a string.

```
string  TypeTimeDescription() const
```

Returned value

Order type by expiration time as a string.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Magic

Gets the ID of expert, that placed the order.

```
long  Magic() const
```

Returned value

ID of expert, that placed the order.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PositionId

Gets the ID of position.

```
long  PositionId() const
```

Returned value

ID of position, in which the order was involved.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

VolumeInitial

Gets the initial volume of order.

```
double  VolumeInitial() const
```

Returned value

Initial volume of order.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

VolumeCurrent

Gets the unfilled volume of order.

```
double  VolumeCurrent() const
```

Returned value

Unfilled volume of order.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceOpen

Gets the order price.

```
double  PriceOpen() const
```

Returned value

Price of order placement.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

StopLoss

Gets the order's Stop Loss.

```
double  StopLoss() const
```

Returned value

Order's Stop Loss.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TakeProfit

Gets the order's Take Profit.

```
double  TakeProfit() const
```

Returned value

Order's Take Profit.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceCurrent

Gets the current price by order symbol.

```
double  PriceCurrent() const
```

Returned value

Current price by order symbol.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceStopLimit

Gets the price of setting limit order.

```
double  PriceStopLimit() const
```

Returned value

Price of setting limit order.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Symbol

Gets the name of order symbol.

```
string  Symbol() const
```

Returned value

Name of order symbol.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Comment

Gets the order comment.

```
string  Comment() const
```

Returned value

Order comment.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_ORDER_PROPERTY_INTEGER  prop_id,     // property ID
   long&                        var          // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of integer type property (value of ENUM_ORDER_PROPERTY_INTEGER enumeration).

var

[out] Reference to long type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  prop_id,      // property ID
   double&                     var          // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property (value of ENUM_ORDER_PROPERTY_DOUBLE enumeration).

var

[out] Reference to double type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_ORDER_PROPERTY_STRING  prop_id,      // property ID
   string&                     var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of text property.

var

[out] Reference to string type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The order should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

StoreState

Saves the order parameters.

```
void  StoreState()
```

Returned value

None.

CheckState

Checks the current parameters against the saved parameters.

```
bool  CheckState()
```

Returned value

true - if the order parameters have changed since the last call of the StoreState() method, otherwise - false.

Select

Selects an order by ticket for further access to its properties.

```
bool  Select(
   ulong  ticket   // order ticket
   )
```

Returned value

true - in case of success, false - if unable to select order.

SelectByIndex

Selects an order by index for further access to its properties.

```
bool  SelectByIndex(
   int  index   // order index
   )
```

Returned value

true - in case of success, false - if unable to select order.

CHistoryOrderInfo

CHistoryOrderInfo is a class for easy access to the history order properties.

Description

CHistoryOrderInfo class provides easy access to the history order properties.

Declaration

```
class CHistoryOrderInfo : public CObject
```

Title

```
#include <Trade\HistoryOrderInfo.mqh>
```

Class methods by groups

Access to integer type properties	
TimeSetup	Gets the time of order placement
TimeSetupMsc	Получает время установки ордера в миллисекундах с 01.01.1970
OrderType	Gets the order type
OrderTypeDescription	Gets the order type as a string
State	Gets the order state
StateDescription	Gets the order state as a string
TimeExpiration	Gets the time of order expiration
TimeDone	Gets the time of order execution or cancellation
TimeDoneMsc	Получает время исполнения или снятия ордера в миллисекундах с 01.01.1970
TypeFilling	Gets the type of order execution by remainder
TypeFillingDescription	Gets the type of order execution by remainder as a string
TypeTime	Gets the type of order at the time of the expiration
TypeTimeDescription	Gets the order type by expiration time as a string
Magic	Gets the ID of expert, that placed the order
PositionId	Gets the ID of position
Access to double type properties	
VolumeInitial	Gets the initial volume of order

VolumeCurrent	Gets the unfilled volume of order
PriceOpen	Gets the order price
StopLoss	Gets the order's Stop Loss
TakeProfit	Gets the order's Take Profit
PriceCurrent	Gets the current price by order symbol
PriceStopLimit	Gets the price of setting limit order
Access to text properties	
Symbol	Gets the order symbol
Comment	Gets the order comment
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets value of specified string type property
Selection	
Ticket	Gets the ticket/selects the order
SelectByIndex	Selects the order by index

TimeSetup

Gets the time of order placement.

```
datetime  TimeSetup() const
```

Returned value

Time of order placement.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeSetupMsc

Получает время установки ордера на исполнение в миллисекундах с 01.01.1970.

```
ulong  TimeSetupMsc() const
```

Возвращаемое значение

Время установки ордера на исполнение в миллисекундах с 01.01.1970.

Примечание

Исторический ордер должен быть предварительно выбран для доступа методом Ticket (по тикету) или SelectByIndex (по индексу).

OrderType

Gets the order type.

```
ENUM_ORDER_TYPE  OrderType() const
```

Returned value

Order type (value of ENUM_ORDER_TYPE enumeration).

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeDescription

Gets the order type as a string.

```
string  TypeDescription() const
```

Returned value

Order type as a string.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

State

Gets the order state.

```
ENUM_ORDER_STATE  State() const
```

Returned value

Order state (value of ENUM_ORDER_STATE enumeration).

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

StateDescription

Gets the order state as a string.

```
string  StateDescription() const
```

Returned value

Order state as a string.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeExpiration

Gets the time of order expiration.

```
datetime  TimeExpiration() const
```

Returned value

Time of order expiration, set on its placement.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeDone

Gets the time of order execution or cancellation.

```
datetime  TimeDone() const
```

Returned value

Time of order execution or cancellation.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeDoneMsc

Получает время исполнения или снятия ордера в миллисекундах с 01.01.1970.

```
ulong  TimeDoneMsc() const
```

Возвращаемое значение

Время исполнения или снятия ордера в миллисекундах с 01.01.1970.

Примечание

Исторический ордер должен быть предварительно выбран для доступа методом Ticket (по тикету) или SelectByIndex (по индексу).

TypeFilling

Gets the type of order execution by remainder.

```
ENUM_ORDER_TYPE_FILLING  TypeFilling() const
```

Returned value

Type of order execution by remainder (value of ENUM_ORDER_TYPE_FILLING enumeration).

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeFillingDescription

Gets the type of order execution by remainder as a string.

```
string  TypeFillingDescription() const
```

Returned value

Type order of execution by remainder as a string.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeTime

Gets the type of order at the time of the expiration.

```
ENUM_ORDER_TYPE_TIME  TypeTime() const
```

Returned value

Type of order at the time of the expiration (value of ENUM_ORDER_TYPE_TIME enumeration).

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeTimeDescription

Gets the order type by expiration time as a string.

```
string  TypeTimeDescription() const
```

Returned value

Order type by expiration time as a string.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Magic

Gets the ID of the Expert Advisor, that placed the order.

```
long  Magic() const
```

Returned value

ID of the Expert Advisor, that placed the order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PositionId

Gets the ID of position.

```
long  PositionId() const
```

Returned value

ID of position, in which the order was involved.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

VolumeInitial

Gets the initial volume of order.

```
double  VolumeInitial() const
```

Returned value

Initial volume of order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

VolumeCurrent

Gets the unfilled volume of order.

```
double  VolumeCurrent() const
```

Returned value

Unfilled volume of order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PriceOpen

Gets the order price.

```
double  PriceOpen() const
```

Returned value

Price of order placement.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

StopLoss

Gets the Stop Loss price of the order.

```
double  StopLoss() const
```

Returned value

Stop Loss price of the order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TakeProfit

Gets the the Take Profit price of the order.

```
double  TakeProfit() const
```

Returned value

The Take Profit price of the order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PriceCurrent

Gets the current price of the order's symbol.

```
double  PriceCurrent() const
```

Returned value

The current price of order's symbol.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PriceStopLimit

Gets the stop limit price of the order.

```
double  PriceStopLimit() const
```

Returned value

Stop Limit price of the order.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Symbol

Gets the name of order symbol.

```
string  Symbol() const
```

Returned value

Name of order symbol.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Comment

Gets the order comment.

```
string  Comment() const
```

Returned value

Order comment.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_ORDER_PROPERTY_INTEGER  prop_id,     // property ID
   long&                        var          // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of integer type property (value of ENUM_ORDER_PROPERTY_INTEGER enumeration).

var

[out] Reference to long type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_ORDER_PROPERTY_DOUBLE  prop_id,      // property ID
   double&                     var          // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property (value of ENUM_ORDER_PROPERTY_DOUBLE enumeration).

var

[out] Reference to double type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_ORDER_PROPERTY_STRING  prop_id,     // property ID
   string&                     var          // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of text property (value of ENUM_ORDER_PROPERTY_STRING enumeration).

var

[out] Reference to string type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The historical order should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Ticket (Get method)

Gets the order ticket.

```
ulong  Ticket() const
```

Returned value

Order ticket.

Ticket (Set method)

Select the order for further work.

```
void  Ticket(
   ulong  ticket      // order ticket
   )
```

Parameters

ticket

[in] Order ticket.

SelectByIndex

Selects an order by index for further access to its properties.

```
bool  SelectByIndex(
   int  index   // order index
   )
```

Returned value

true - in case of success, false - if unable to select order.

CPositionInfo

CPositionInfo is a class for easy access to the open position properties.

Description

CPositionInfo class provides easy access to the open position properties.

Declaration

```
class CPositionInfo : public CObject
```

Title

```
#include <Trade\PositionInfo.mqh>
```

Class methods by groups

Access to integer type properties	
Time	Gets the time of position opening
TimeMsc	Получает время открытия позиции в миллисекундах с 01.01.1970
TimeUpdate	Получает время изменения позиции в секундах с 01.01.1970
TimeUpdateMsc	Получает время изменения позиции в миллисекундах с 01.01.1970
PositionType	Gets the position type
TypeDescription	Gets the position type as a string
Magic	Gets the ID of expert, that opened the position
Identifier	Gets the ID of position
Access to double type properties	
Volume	Gets the volume of position
PriceOpen	Gets the price of position opening
StopLoss	Gets the price of position's Stop Loss
TakeProfit	Gets the price of position's Take Profit
PriceCurrent	Gets the current price by position symbol
Commission	Gets the amount of commission by position
Swap	Gets the amount of swap by position
Profit	Gets the amount of current profit by position
Access to text properties	

Symbol	Gets the name of position symbol
Comment	Gets the comment of the position
Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets the value of specified string type property
Selection	
Select	Selects the position
SelectByIndex	Selects the position by index
State	
StoreState	Saves the position parameters
CheckState	Checks the current parameters against the saved parameters

Time

Gets the time of position opening.

```
datetime  Time() const
```

Returned value

Time of position opening.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TimeMsc

Получает время открытия позиции в миллисекундах с 01.01.1970.

```
ulong  TimeMsc() const
```

Возвращаемое значение

Время открытия позиции в миллисекундах с 01.01.1970.

Примечание

Позиция должна быть предварительно выбрана для доступа методом Select (по символу) или SelectByIndex (по индексу).

TimeUpdate

Получает время изменения позиции в секундах с 01.01.1970.

```
datetime  TimeUpdate() const
```

Возвращаемое значение

Время изменения позиции в секундах с 01.01.1970.

Примечание

Позиция должна быть предварительно выбрана для доступа методом Select (по символу) или SelectByIndex (по индексу).

TimeUpdateMsc

Получает время изменения позиции в миллисекундах с 01.01.1970.

```
ulong  TimeUpdateMsc() const
```

Возвращаемое значение

Время изменения позиции в миллисекундах с 01.01.1970.

Примечание

Позиция должна быть предварительно выбрана для доступа методом Select (по символу) или SelectByIndex (по индексу).

PositionType

Gets the position type.

```
ENUM_POSITION_TYPE  PositionType() const
```

Returned value

Position type (value of ENUM_POSITION_TYPE enumeration).

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TypeDescription

Gets the position type as a string.

```
string  TypeDescription() const
```

Returned value

Position type as a string.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Magic

Gets the ID of Expert Advisor, opened the position.

```
long  Magic() const
```

Returned value

ID of the Expert Advisor, opened the position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Identifier

Gets the ID of position.

```
long  Identifier() const
```

Returned value

ID of position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Volume

Gets the volume of position.

```
double  Volume() const
```

Returned value

Volume of position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceOpen

Gets the price of position opening.

```
double  PriceOpen() const
```

Returned value

Position open price.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

StopLoss

Gets the Stop Loss price of the position.

```
double  StopLoss() const
```

Returned value

The Stop Loss price of the position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

TakeProfit

Gets the Take Profit price of the position.

```
double  TakeProfit() const
```

Returned value

The Take Profit price of the position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

PriceCurrent

Gets the current price by position symbol.

```
double  PriceCurrent() const
```

Returned value

Current price by position symbol.

Commission

Gets the amount of commission of the position.

```
double  Commission() const
```

Returned value

Amount of commission of the position (in deposit currency).

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Swap

Gets the amount of swap of the position.

```
double  Swap() const
```

Returned value

Amount of swap of the position (in deposit currency).

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Profit

Gets the amount of current profit of the position.

```
double  Profit() const
```

Returned value

Amount of current profit of the position (in deposit currency).

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Symbol

Gets the name of position symbol.

```
string  Symbol() const
```

Returned value

Name of position symbol.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Comment

Gets the comment of the postion.

```
string  Comment() const
```

Returned value

Comment of the position.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_POSITION_PROPERTY_INTEGER  prop_id,      // property ID
   long&                           var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of integer type property (value of ENUM_POSITION_PROPERTY_INTEGER enumeration).

var

[out] Reference to long type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_POSITION_PROPERTY_DOUBLE  prop_id,      // property ID
   double&                        var          // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property (value of ENUM_POSITION_PROPERTY_DOUBLE enumeration).

var

[in] Reference to double type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_POSITION_PROPERTY_STRING  prop_id,      // property ID
   string&                        var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of text property (value of ENUM_POSITION_PROPERTY_STRING enumeration).

var

[out] Reference to string type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The position should be selected using the Select (by ticket) or SelectByIndex (by index) methods.

Select

Select the position for further work.

```
bool  Select(
   const string  symbol      // symbol
   )
```

Parameters

symbol

[in] Symbol for position selection.

SelectByIndex

Selects the position by index for further access to its properties.

```
bool  SelectByIndex(
   int  index   // position index
   )
```

Returned value

true - in case of success, false - if unable to select position.

StoreState

Saves the position parameters.

```
void  StoreState()
```

Returned value

None.

CheckState

Checks the current parameters against the saved parameters.

```
bool  CheckState()
```

Returned value

true - if the position parameters have changed since the last call of the StoreState() method, otherwise - false.

CDealInfo

CDealInfo is a class for easy access to the deal properties.

Description

CDealInfo class provides access to the deal properties.

Declaration

```
class CDealInfo : public CObject
```

Title

```
#include <Trade\DealInfo.mqh>
```

Class methods by groups

Access to integer type properties	
Order	Gets the order by which the deal is executed
Time	Gets the time of deal execution
TimeMsc	Получает время совершения сделки в миллисекундах с 01.01.1970
DealType	Gets the deal type
TypeDescription	Gets the deal type as a string
Entry	Gets the deal direction
EntryDescription	Gets the deal direction as a string
Magic	Gets the ID of expert, that executed the deal
PositionId	Gets the ID of position, in which the deal was involved
Access to double type properties	
Volume	Gets the volume of deal
Price	Gets the deal price
Commision	Gets the amount of commission by deal
Swap	Gets the amount of swap when position is closed
Profit	Gets the financial result of deal
Access to text properties	
Symbol	Gets the name of deal symbol
Comment	Gets the deal comment

Access to MQL5 API functions	
InfoInteger	Gets the value of specified integer type property
InfoDouble	Gets the value of specified double type property
InfoString	Gets value of specified string type property
Selection	
Ticket	Gets ticket/selects the deal
SelectByIndex	Selects the deal by index

Order

Gets the order by which the deal is executed.

```
long  Order() const
```

Returned value

Order by which the deal is executed.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Time

Gets the time of deal execution.

```
datetime  Time() const
```

Returned value

Time of deal execution.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TimeMsc

Получает время совершения сделки в миллисекундах с 01.01.1970.

```
ulong  TimeMsc() const
```

Возвращаемое значение

Время совершения сделки в миллисекундах с 01.01.1970.

Примечание

Сделка должна быть предварительно выбрана для доступа методом Ticket (по тикету) или SelectByIndex (по индексу).

DealType

Gets the deal type.

```
ENUM_DEAL_TYPE  DealType() const
```

Returned value

Deal type (value of ENUM_DEAL_TYPE enumeration).

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

TypeDescription

Gets the deal type as a string.

```
string  TypeDescription() const
```

Returned value

Deal type as a string.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Entry

Gets the deal direction.

```
ENUM_DEAL_ENTRY  Entry() const
```

Returned value

Deal direction (value of ENUM_DEAL_ENTRY enumeration.).

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

EntryDescription

Gets the deal direction as a string.

```
string  EntryDescription() const
```

Returned value

Deal direction as a string.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Magic

Gets the ID of the Expert Advisor, that executed the deal.

```
long  Magic() const
```

Returned value

ID of the Expert Advisor, that executed the deal.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

PositionId

Gets the ID of position, in which the deal was involved.

```
long  PositionId() const
```

Returned value

ID of position, in which the deal was involved.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Volume

Gets the volume of deal.

```
double  Volume() const
```

Returned value

Volume of deal.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Price

Gets the deal price.

```
double  Price() const
```

Returned value

Deal price.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Commission

Gets the amount of commission of the deal.

```
double  Commission() const
```

Returned value

Amount of commission of the deal.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Swap

Gets the amount of swap when position is closed.

```
double  Swap() const
```

Returned value

Amount of swap when position is closed.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Profit

Gets the financial result of the deal.

```
double  Profit() const
```

Returned value

Financial result of the deal (in deposit currency).

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Symbol

Gets the name of the deal symbol.

```
string  Symbol() const
```

Returned value

Name of the deal symbol.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Comment

Gets the deal comment.

```
string  Comment() const
```

Returned value

Deal comment.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoInteger

Gets the value of specified integer type property.

```
bool  InfoInteger(
   ENUM_DEAL_PROPERTY_INTEGER  prop_id,      // property ID
   long&                       var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of integer type property (value of ENUM_DEAL_PROPERTY_INTEGER enumeration).

var

[out] Reference to long type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoDouble

Gets the value of specified double type property.

```
bool  InfoDouble(
   ENUM_DEAL_PROPERTY_DOUBLE  prop_id,      // property ID
   double&                    var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of double type property (value of ENUM_DEAL_PROPERTY_DOUBLE enumeration).

var

[in] Reference to double type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

InfoString

Gets the value of specified string type property.

```
bool  InfoString(
   ENUM_DEAL_PROPERTY_STRING  prop_id,      // property ID
   string&                    var           // reference to variable
   ) const
```

Parameters

prop_id

[in] ID of text property (value of ENUM_DEAL_PROPERTY_STRING enumeration).

var

[out] Reference to string type variable to place result.

Returned value

true - in case of success, false - if unable to get property value.

Note

The deal should be selected using the Ticket (by ticket) or SelectByIndex (by index) methods.

Ticket (Get method)

Gets the deal ticket.

```
ulong  Ticket() const
```

Returned value

Deal ticket.

Ticket (Set method)

Select the position for further work.

```
void  Ticket(
   ulong  ticket      // ticket
   )
```

Parameters

ticket

[in] Deal ticket.

SelectByIndex

Selects the deal by index for further access to its properties.

```
bool  SelectByIndex(
   int  index   // order index
   )
```

Returned value

true - in case of success, false - if unable to select the deal.

CTrade

CTrade is a class for easy access to the trade functions.

Description

CTrade class provides easy access to the trade functions.

Declaration

```
class CTrade : public CObject
```

Title

```
#include <Trade\Trade.mqh>
```

Class methods by groups

Setting parameters	
LogLevel	Sets logging level
SetExpertMagicNumber	Sets the expert ID
SetDeviationInPoints	Sets the allowed deviation
SetTypeFilling	Sets filling type of the order
SetAsyncMode	Sets asynchronous mode for trade operations
Operations with orders	
OrderOpen	Places the pending order with set parameters
OrderModify	Modifies the pending order parameters
OrderDelete	Deletes the pending order
Operations with positions	
PositionOpen	Opens the position with set parameters
PositionModify	Modifies the position parameters
PositionClose	Closes the position
Additional methods	
Buy	Opens a long position with specified parameters
Sell	Opens a short position with specified parameters
BuyLimit	Places the pending order of Buy Limit type with specified parameters
BuyStop	Places the pending order of Buy Stop type with

	specified parameters
SellLimit	Places the pending order of Sell Limit type with specified parameters
SellStop	Places the pending order of Sell Stop type with specified parameters
Access to the last request parameters	
Request	Gets the copy of the last request structure
RequestAction	Gets the trade operation type
RequestActionDescription	Gets the trade operation type as string
RequestMagic	Gets the magic number of the Expert Advisor
RequestOrder	Gets the order ticket, used in the last request
RequestSymbol	Gets the name of the symbol, used in the last request
RequestVolume	Gets the trade volume (in lots), used in the last request
RequestPrice	Gets the price, used in the last request
RequestStopLimit	Gets the price of pending order of Stop Limit type, used in the last request
RequestSL	Gets the Stop Loss price of the order, used in the last request
RequestTP	Gets the Take Profit price of the order, used in the last request
RequestDeviation	Gets the price deviation of the order, used in the last request
RequestType	Gets the type of the order, used in the last request
RequestTypeDescription	Gets the type of the order (as string) , used in the last request
RequestTypeFilling	Gets the filling type of the order, used in the last request
RequestTypeFillingDescription	Gets the filling type of the order (as string), used in the last request
RequestTypeTime	Gets the validity period of the order, used in the last request
RequestTypeTimeDescription	Gets the validity period of the order (as string), used in the last request
RequestExpiration	Gets the expiration time of the order, used in the last request

RequestComment	Gets the comment of the order, used in the last request
Access to the last request checking results	
CheckResult	Gets the copy of the structure of the last request check result.
CheckResultRetcode	Gets the value of the retcode field of MqlTradeCheckResult type, filled while checking of the request correctness
CheckResultRetcodeDescription	Gets the string description of the retcode field of MqlTradeCheckResult type, filled while checking of the request correctness
CheckResultBalance	Gets the value of the balance field of MqlTradeCheckResult type, filled while checking of the request correctness
CheckResultEquity	Gets the value of the equity field of MqlTradeCheckResult type, filled while checking of the request correctness
CheckResultProfit	Gets the value of the profit field of MqlTradeCheckResult type, filled while checking of the request correctness
CheckResultMargin	Gets the value of the margin field of MqlTradeCheckResult type, filled while checking of the request correctness
CheckResultMarginFree	Gets the value of the margin_free field of MqlTradeCheckResult type, filled while checking of the request correctness
CheckResultMarginLevel	Gets the value of the margin_level field of MqlTradeCheckResult type, filled while checking of the request correctness
CheckResultComment	Gets the value of the comment field of MqlTradeCheckResult type, filled while checking of the request correctness
Access to the last request execution results	
Result	Gets the copy of the structure of the last request result
ResultRetcode	Gets the code of request result
ResultRetcodeDescription	Gets the code of request result as text
ResultDeal	Gets the deal ticket
ResultOrder	Gets the order ticket
ResultVolume	Gets the volume of deal or order

ResultPrice	Gets the price, confirmed by broker
ResultBid	Gets the current bid price (the requote)
ResultAsk	Gets the current ask price (the requote)
ResultComment	Gets the broker comment
Auxiliary methods	
PrintRequest	Prints the last request parameters into journal
PrintResult	Prints the results of the last request into journal
FormatRequest	Prepares the formatted string with last request parameters
FormatRequestResult	Prepares the formatted string with results of the last request execution

LogLevel

Sets logging level for messages.

```
void  LogLevel(
   int  log_level      // logging level
   )
```

Parameters

log_level

[in] Logging level.

Returned value

None.

Note

Log_level = 0 - logging disabled (used in optimization mode).
Log_level = 1 - logging error messages (default).
Log_level = 2 - logging all messages (used in testing mode).

SetExpertMagicNumber

Sets the expert ID.

```
void  SetExpertMagicNumber(
   ulong  magic      // ID
   )
```

Parameters

magic

[in] New ID of the expert.

Returned value

None.

SetDeviationInPoints

Sets the allowed deviation.

```
void  SetDeviationInPoints(
   ulong  deviation      // deviation
   )
```

Parameters

deviation

[in] Allowed deviation.

Returned value

None.

SetTypeFilling

Sets filling type of the order.

```
void  SetTypeFilling(
   ENUM_ORDER_TYPE_FILLING  filling      // order filling type
   )
```

Parameters

filling

[in] Order filling type (value of ENUM_ORDER_TYPE_FILLING enumeration).

Returned value

None.

OrderOpen

Places the pending order with set parameters.

```
bool  OrderOpen(
   const string          symbol,          // symbol
   ENUM_ORDER_TYPE       order_type,      // order type
   double                volume,          // order volume
   double                limit_price,     // StopLimit price
   double                price,           // execution price
   double                sl,              // Stop Loss price
   double                tp,              // Take Profit price
   ENUM_ORDER_TYPE_TIME  type_time,       // type by expiration
   datetime              expiration,      // expiration
   const string          comment=""       // comment
   )
```

Parameters

symbol

[in] Name of trade instrument.

order_type

[in] Type of order trade operation (value of ENUM_ORDER_TYPE enumeration).

volume

[in] Requested order volume.

limit_price

[in] Price at which the StopLimit order will be placed.

price

[in] Price at which the order must be executed.

sl

[in] Price at which the Stop Loss will trigger.

tp

[in] Price at which the Take Profit will trigger.

type_time

[in] Order type by execution (value of ENUM_ORDER_TYPE_TIME enumeration).

expiration

[in] Expiration date of pending order.

comment=""

[in] Order comment.

Returned value

true - in case of successful check of the basic structures, otherwise - false.

Note

Successful completion of the OrderSend(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode() and value, returned by ResultOrder().

OrderModify

Modifies the pending order parameters.

```
bool  OrderModify(
   ulong                ticket,         // order ticket
   double               price,          // execution price
   double               sl,             // Stop Loss price
   double               tp,             // Take Profit price
   ENUM_ORDER_TYPE_TIME type_time,      // type by expiration
   datetime             expiration,     // expiration
   double               stoplimit       // цена Limit ордера
   )
```

Parameters

ticket

[in] Order ticket.

price

[in] The new price by which the order must be executed (or the previous value, if the change is not necessary).

sl

[in] The new price by which the Stop Loss will trigger (or the previous value, if the change is not necessary).

tp

[in] The new price by which the Take Profit will trigger (or the previous value, if the change is not necessary).

type_time

[in] The new type of order by expiration (or the previous value, if the change is not necessary), value of ENUM_ORDER_TYPE_TIME enumeration.

expiration

[in] The new expiration date of pending order (or the previous value, if the change is not necessary).

stoplimit

[in] Новая цена, по которой будет выставлен Limit ордер при достижении ценой значения *price*. Указывается только для StopLimit ордеров.

Returned value

true - in case of successful check of the basic structures, otherwise - false.

Note

Successful completion of the OrderModify(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode().

OrderDelete

Deletes the pending order.

```
bool  OrderDelete(
   ulong  ticket      // order ticket
   )
```

Parameters

ticket

[in] Order ticket.

Returned value

true - in case of successful check of the basic structures, otherwise - false.

Note

Successful completion of the OrderDelete(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode().

PositionOpen

Opens the position with set parameters.

```
bool  PositionOpen(
   const string      symbol,          // symbol
   ENUM_ORDER_TYPE   order_type,      // position type
   double            volume,          // position volume
   double            price,           // execution price
   double            sl,              // Stop Loss price
   double            tp,              // Take Profit price
   const string      comment=""       // comment
   )
```

Parameters

symbol

[in] Name of trade instrument, by which it is intended to open position.

order_type

[in] Type of position trade operation (value of ENUM_ORDER_TYPE enumeration).

volume

[in] Requested position volume.

price

[in] Price at which the position must be opened.

sl

[in] Price at which the Stop Loss will trigger.

tp

[in] Price at which the Take Profit will trigger.

comment=""

[in] Position comment.

Returned value

true - in case of successful check of the basic structures, otherwise - false.

Note

Successful completion of the PositionOpen(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode() and value, returned by ResultDeal().

PositionModify

Modifies the position parameters by specified symbol.

```
bool  PositionModify(
   const string  symbol,      // symbol
   double        sl,          // Stop Loss price
   double        tp           // Take Profit price
   )
```

Parameters

symbol

[in] Name of trade instrument, by which it is intended to modify position.

sl

[in] The new price by which the Stop Loss will trigger (or the previous value, if the change is not necessary).

tp

[in] The new price by which the Take Profit will trigger (or the previous value, if the change is not necessary).

Returned value

true - in case of successful check of the basic structures, otherwise - false.

Note

Successful completion of the PositionModify(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode().

PositionClose

Closes the position by specified symbol.

```
bool  PositionClose(
   const string  symbol,                  // symbol
   ulong         deviation=ULONG_MAX      // deviation
   )
```

Parameters

symbol

[in] Name of trade instrument, by which it is intended to close position.

deviation=ULONG_MAX

[in] Maximal deviation from the current price (in points).

Returned value

true - in case of successful check of the basic structures, otherwise - false.

Note

Successful completion of the PositionClose(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode().

Buy

Opens a long position with specified parameters.

```
bool  Buy(
   double         volume,           // position volume
   const string   symbol=NULL,      // symbol
   double         price=0.0,        // price
   double         sl=0.0,           // stop loss price
   double         tp=0.0,           // take profit price
   const string   comment=""        // comment
   )
```

Parameters

volume

[in] Position volume.

symbol=NULL

[in] Position symbol. If the symbol isn't specified, the current symbol will be used.

price=0.0

[in] Price. If the price isn't specified, the current market Ask price will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

comment=""

[in] Comment.

Returned value

true - in case of successful check of the structures, otherwise - false.

Note

Successful completion of the Buy(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode() and value, returned by ResultDeal().

Sell

Opens a short position with specified parameters.

```
bool  Sell(
   double        volume,          // position volume
   const string  symbol=NULL,     // symbol
   double        price=0.0,       // price
   double        sl=0.0,          // stop loss price
   double        tp=0.0,          // take profit price
   const string  comment=""       // comment
   )
```

Parameters

volume

[in] Position volume.

symbol=NULL

[in] Position symbol. If the symbol isn't specified, the current symbol will be used.

price=0.0

[in] Price. If the price isn't specified, the current market Bid price will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

comment=""

[in] Comment.

Returned value

true - in case of successful check of the structures, otherwise - false.

Note

Successful completion of the Sell(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode() and value, returned by ResultDeal().

BuyLimit

Places the pending order of Buy Limit type (buy at the price, lower than current market price) with specified parameters.

```
bool  BuyLimit(
   double                volume,                    // order volume
   double                price,                     // order price
   const string          symbol=NULL,               // symbol
   double                sl=0.0,                    // stop loss price
   double                tp=0.0,                    // take profit price
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,   // order lifetime
   datetime              expiration=0,              // order expiration time
   const string          comment=""                 // comment
   )
```

Parameters

volume

[in] Order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol isn't specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime (value of ENUM_ORDER_TYPE_TIME enumeration).

expiration=0

[in] Order expiration time (used only if type_time=ORDER_TIME_SPECIFIED).

comment=""

[in] Order comment.

Returned value

true - in case of successful check of the structures, otherwise - false.

Note

Successful completion of the BuyLimit(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode() and value, returned by ResultDeal().

BuyStop

Places the pending order of Buy Stop type (buy at the price, higher than current market price) with specified parameters.

```
bool  BuyStop(
   double                volume,                    // order volume
   double                price,                     // order price
   const string          symbol=NULL,               // symbol
   double                sl=0.0,                    // stop loss price
   double                tp=0.0,                    // take profit price
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,   // order lifetime
   datetime              expiration=0,              // order expiration time
   const string          comment=""                 // comment
   )
```

Parameters

volume

[in] Order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol isn't specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime (value of ENUM_ORDER_TYPE_TIME enumeration).

expiration=0

[in] Order expiration time (used only if type_time=ORDER_TIME_SPECIFIED).

comment=""

[in] Order comment.

Returned value

true - in case of successful check of the structures, otherwise - false.

Note

Successful completion of the BuyStop(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode() and value, returned by ResultDeal().

SellLimit

Places the pending order of Sell Limit type (sell at the price, higher than current market price) with specified parameters.

```
bool  SellLimit(
   double                volume,                    // order volume
   double                price,                     // order price
   const string          symbol=NULL,               // symbol
   double                sl=0.0,                    // stop loss price
   double                tp=0.0,                    // take profit price
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,   // order lifetime
   datetime              expiration=0,              // order expiration time
   const string          comment=""                 // comment
   )
```

Parameters

volume

[in] Order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol isn't specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime (value of ENUM_ORDER_TYPE_TIME enumeration).

expiration=0

[in] Order expiration time (used only if type_time=ORDER_TIME_SPECIFIED).

comment=""

[in] Order comment.

Returned value

true - in case of successful check of the structures, otherwise - false.

Note

Successful completion of the SellLimit(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode() and value, returned by ResultDeal().

SellStop

Places the pending order of Buy Stop type (sell at the price, lower than current market price) with specified parameters.

```
bool  SellStop(
   double                volume,                    // order volume
   double                price,                     // order price
   const string          symbol=NULL,               // symbol
   double                sl=0.0,                    // stop loss price
   double                tp=0.0,                    // take profit price
   ENUM_ORDER_TYPE_TIME  type_time=ORDER_TIME_GTC,   // order lifetime
   datetime              expiration=0,              // order expiration time
   const string          comment=""                 // comment
   )
```

Parameters

volume

[in] Order volume.

price

[in] Order price.

symbol=NULL

[in] Order symbol. If the symbol isn't specified, the current symbol will be used.

sl=0.0

[in] Stop Loss price.

tp=0.0

[in] Take Profit price.

type_time=ORDER_TIME_GTC

[in] Order lifetime (value of ENUM_ORDER_TYPE_TIME enumeration).

expiration=0

[in] Order expiration time (used only if type_time=ORDER_TIME_SPECIFIED).

comment=""

[in] Order comment.

Returned value

true - in case of successful check of the structures, otherwise - false.

Note

Successful completion of the SellStop(...) method does not always mean successful execution of the trade operation. It's necessary to check the result of trade request (trade server return code) using ResultRetcode() and value, returned by ResultDeal().

Request

Gets the copy of the last request structure.

```
void  Request(
   MqlTradeRequest&  request      // target structure
   ) const
```

Parameters

request

[out] Reference to the structure of MqlTradeRequest type.

Returned value

None.

RequestAction

Gets the trade operation type.

```
ENUM_TRADE_REQUEST_ACTIONS  RequestAction() const
```

Returned value

Trade operation type, used in the last request.

RequestActionDescription

Gets the trade operation type as string.

```
string  RequestActionDescription() const
```

Returned value

Trade operation type (as string), used in the last request.

RequestMagic

Gets the magic number of the Expert Advisor.

```
ulong  RequestMagic() const
```

Returned value

The magic number (ID) of the Expert Advisor, used in the last request.

RequestOrder

Gets the order ticket, used in the last request.

```
ulong  RequestOrder() const
```

Returned value

Order ticket of the last request.

RequestSymbol

Gets the name of the symbol, used in the last request.

```
string  RequestSymbol() const
```

Returned value

The name of the symbol, used in the last request.

RequestVolume

Gets the trade volume (in lots), used in the last request.

```
double  RequestVolume() const
```

Returned value

The trade volume (in lots), used in the last request.

RequestPrice

Gets the price, used in the last request.

```
double  RequestPrice() const
```

Returned value

Order price, used in the last request.

RequestStopLimit

Gets the price of pending order of Stop Limit type, used in the last request.

```
double  RequestStoplimit() const
```

Returned value

The price of pending order of Stop Limit type, used in the last request.

RequestSL

Gets the Stop Loss price of the order, used in the last request.

```
double  RequestSL() const
```

Returned value

The Stop Loss price, used in the last request.

RequestTP

Gets the Take Profit price of the order, used in the last request.

```
double  RequestTP() const
```

Returned value

The Take Profit price, used in the last request.

RequestDeviation

Gets the price deviation of the order, used in the last request.

```
ulong  RequestDeviation() const
```

Returned value

The price deviation of the order, used in the last request.

RequestType

Gets the type of the order, used in the last request.

```
ENUM_ORDER_TYPE  RequestType() const
```

Returned value

Order type, used in the last request (value of ENUM_ORDER_TYPE enumeration).

RequestTypeDescription

Gets the type of the order (as string) , used in the last request.

```
string  RequestTypeDescription() const
```

Returned value

The order type (as string), used in the last request.

RequestTypeFilling

Gets the filling type of the order, used in the last request.

```
ENUM_ORDER_TYPE_FILLING  RequestTypeFilling() const
```

Returned value

The filling type of the order (value of ENUM_ORDER_TYPE_FILLING), used in the last request.

RequestTypeFillingDescription

Gets the filling type of the order (as string), used in the last request.

```
string  RequestTypeFillingDescription() const
```

Returned value

The filling type (as string) of the order, used in the last request.

RequestTypeTime

Gets the validity period of the order, used in the last request.

```
ENUM_ORDER_TYPE_TIME  RequestTypeTime() const
```

Returned value

The validity period of the order (value of ENUM_ORDER_TYPE_TIME enumeration), used in the last request.

RequestTypeTimeDescription

Gets the validity period of the order (as string), used in the last request.

```
string  RequestTypeTimeDescription() const
```

Returned value

The validity period of the order (as string), used in the last request.

RequestExpiration

Gets the expiration time of the order, used in the last request.

```
datetime  RequestExpiration() const
```

Returned value

The expiration time of the order, used in the last request.

RequestComment

Gets the comment of the order, used in the last request.

```
string  RequestComment() const
```

Returned value

The comment of the order, used in the last request.

Result

Gets the copy of the structure of the last request result.

```
void  Result(
   MqlTradeResult&  result      // reference
   ) const
```

Parameters

result

[out] Reference to the structure of MqlTradeRequest type.

Returned value

None.

ResultRetcode

Gets the code of request result.

```
uint  ResultRetcode() const
```

Returned value

The Code of request result.

ResultRetcodeDescription

Gets the code of request result as text.

```
string  ResultRetcodeDescription() const
```

Returned value

Code of the last request result as text.

ResultDeal

Gets the deal ticket.

```
ulong  ResultDeal() const
```

Returned value

Deal ticket, if the deal is executed.

ResultOrder

Gets the order ticket.

```
ulong  ResultOrder() const
```

Returned value

Order ticket, if the order is placed.

ResultVolume

Gets the volume of deal or order.

```
double  ResultVolume() const
```

Returned value

Volume of deal or order.

ResultPrice

Gets the price, confirmed by broker.

```
double  ResultPrice() const
```

Returned value

Price, confirmed by the broker.

ResultBid

Gets the current bid price (the requote).

```
double  ResultBid() const
```

Returned value

Current bid price (the requote).

ResultAsk

Gets the current ask price (the requote).

```
double  ResultAsk() const
```

Returned value

Current ask price (the requote).

ResultComment

Gets the broker comment.

```
string  ResultComment() const
```

Returned value

Broker comment to the operation.

CheckResult

Gets the copy of the structure of the last request check result.

```
void  CheckResult(
   MqlTradeCheckResult&  check_result      // reference
   ) const
```

Parameters

check_result

[out] Reference to the target structure of the MqlTradeCheckResult type.

Returned value

None.

CheckResultRetcode

Gets the value of the retcode field of MqlTradeCheckResult type, filled while checking of the request correctness.

```
uint  CheckResultRetcode() const
```

Returned value

The value of the retcode field (error code) of MqlTradeCheckResult type, filled while checking of the request correctness.

CheckResultRetcodeDescription

Gets the string description of the retcode field of MqlTradeCheckResult type, filled while checking of the request correctness.

```
string  ResultRetcodeDescription() const
```

Returned value

The string description of the retcode field (Error code) of MqlTradeCheckResult type, filled while checking of the request correctness.

CheckResultBalance

Gets the value of the balance field of MqlTradeCheckResult type, filled while checking of the request correctness.

```
double  CheckResultBalance() const
```

Returned value

The value of the balance field (balance value that will be after the execution of the trade operation) of MqlTradeCheckResult type, filled while checking of the request correctness.

CheckResultEquity

Gets the value of the equity field of MqlTradeCheckResult type, filled while checking of the request correctness.

```
double  CheckResultEquity() const
```

Returned value

The value of the equity field (equity value that will be after the execution of the trade operation) of MqlTradeCheckResult type, filled while checking of the request correctness.

CheckResultProfit

Gets the value of the profit field of MqlTradeCheckResult type, filled while checking of the request correctness.

```
double  CheckResultProfit() const
```

Returned value

The value of the profit field (profit value that will be after the execution of the trade operation) of MqlTradeCheckResult type, filled while checking of the request correctness.

CheckResultMargin

Gets the value of the margin field of MqlTradeCheckResult type, filled while checking of the request correctness.

```
double  CheckResultMargin() const
```

Returned value

The value of the margin field (margin required for the trade operation) of MqlTradeCheckResult type, filled while checking of the request correctness.

CheckResultMarginFree

Gets the value of the margin_free field of MqlTradeCheckResult type, filled while checking of the request correctness.

```
double  CheckResultMarginFree() const
```

Returned value

The value of the margin_free field (free margin that will be left after the execution of the trade operation) of MqlTradeCheckResult type, filled while checking of the request correctness.

CheckResultMarginLevel

Gets the value of the margin_level field of MqlTradeCheckResult type, filled while checking of the request correctness.

```
double  CheckResultMarginLevel() const
```

Returned value

The value of the margin_level field (margin level that will be set after the execution of the trade operation) of MqlTradeCheckResult type, filled while checking of the request correctness.

CheckResultComment

The value of the comment field of MqlTradeCheckResult type, filled while checking of the request correctness.

```
string  CheckResultComment() const
```

Returned value

The value of the comment field (Comment to the reply code, error description) of MqlTradeCheckResult type, filled while checking of the request correctness.

PrintRequest

Prints the last request parameters into journal.

```
void  PrintRequest() const
```

Returned value

None.

PrintResult

Prints the results of the last request into journal.

```
void  PrintResult() const
```

Returned value

None.

FormatRequest

Prepares the formatted string with last request parameters.

```
string  FormatRequest(
   string&                 str,          // target string
   const MqlTradeRequest&  request       // request
   ) const
```

Parameters

str

[in] Target string, passed by reference.

request

[in] A structure of MqlTradeRequest type with parameters of the last request.

Returned value

None.

FormatRequestResult

Prepares the formatted string with results of the last request execution.

```
string  FormatRequestResult(
   string&                 str,          // string
   const MqlTradeRequest&  request,      // request structure
   const MqlTradeResult&   result        // result structure
   ) const
```

Parameters

str

[in] Target string, passed by reference.

request

[in] A structure of MqlTradeRequest type with parameters of the last request.

result

[in] A structure of MqlTradeResult type with results of the last request.

Returned value

None.

CTerminalInfo

CTerminalInfo is a class for simplified access to the properties of mql5 program environment.

Description

CTerminalInfo class provides access to the properties of mql5 program environment.

Declaration

```
class CTerminalInfo : public CObject
```

Title

```
#include <Trade\TerminalInfo.mqh>
```

Class methods by groups

Methods for access to the properties of integer type	
Build	Gets the build number of the client terminal
IsConnected	Gets the information about connection to trade server
IsDLLsAllowed	Gets the information about permission of DLL usage
IsTradeAllowed	Gets the information about permission to trade
IsEmailEnabled	Gets the information about permission to send e-mails to SMTP-server and login, specified in the terminal settings
IsFtpEnabled	Gets the information about permission to send trade reports to FTP server and login, specified in the terminal settings
MaxBars	Gets the information about maximum number of bars on chart
CodePage	Gets the information about the code page of the language in the client terminal
CPUCores	Gets the information about the CPU cores
MemoryPhysical	Gets the information about the physical memory (in Mb)
MemoryTotal	Gets the information about the total memory, available for the terminal/agent process (in Mb)
MemoryAvailable	Gets the information about the free memory, available for the terminal/agent process (in

	Mb)
MemoryUsed	Gets the information about the memory, used by the terminal/agent process (in Mb)
IsX64	Gets the information about the type of the client terminal (32/64 bit)
OpenCLSupport	Gets the information about the version of OpenCL, supported by video card
DiskSpace	Gets the information about free disk space (in Mb)
Methods for access to the properties of string type	
Language	Gets the language of the client terminal
Name	Gets the name of the client terminal
Company	Gets the company of the client terminal
Path	Gets the folder of the client terminal
DataPath	Gets the data folder of the client terminal
CommonDataPath	Gets the common data folder of all client terminals, installed on the computer
Access to MQL5 API functions	
InfoInteger	Gets the value of the property of integer type
InfoString	Gets the value of property of string type

Build

Gets the build number of the client terminal.

```
int  CBuild() const
```

Returned value

Build number of the client terminal.

Note

To get the build number it uses the TerminalInfoInteger() function (TERMINAL_BUILD property).

IsConnected

Gets the information about connection to trade server.

```
bool  IsConnected() const
```

Returned value

true, if terminal is connected to trade server, otherwise false.

Note

To get connection status it uses the TerminalInfoInteger() function (TERMINAL_CONNECTED property).

IsDLLsAllowed

Gets the information about permission of DLL usage.

```
bool  IsDLLsAllowed() const
```

Returned value

true, if DLL usage is allowed, otherwise false.

Note

To get permission of DLL usage it uses the TerminalInfoInteger() function (TERMINAL_DLLS_ALLOWED property).

IsTradeAllowed

Gets the information about permission to trade.

```
bool  IsTradeAllowed() const
```

Returned value

true, if trade allowed, otherwise false.

Note

To get permission to trade it uses the TerminalInfoInteger() function (TERMINAL_TRADE_ALLOWED property).

IsEmailEnabled

Gets the information about permission to send e-mails to SMTP-server and login, specified in the terminal settings.

```
bool  IsEmailEnabled() const
```

Returned value

true, if sending of e-mails is allowed, otherwise false.

Note

To get the permission of e-mails sending it uses the TerminalInfoInteger() function (TERMINAL_EMAIL_ENABLED property).

IsFtpEnabled

Gets the information about permission to send trade reports to FTP server and login, specified in the terminal settings.

```
bool  IsFtpEnabled() const
```

Returned value

true, if trade reports sending to FTP server is allowed, otherwise false.

Note

To get the information about permission to send trade reports it uses the TerminalInfoInteger() function (TERMINAL_FTP_ENABLED property).

MaxBars

Gets the maximum number of bars on chart, specified in the client terminal settings.

```
int  MaxBars() const
```

Returned value

Maximum number of bars on the chart.

Note

To get the maximum number of bars on the chart it uses the TerminalInfoInteger() function (TERMINAL_MAXBARS property).

CodePage

Gets the information about code page of the language in the client terminal.

```
int  CodePage() const
```

Returned value

Code page of the language in the client terminal.

Note

To get the code page it uses the TerminalInfoInteger() function (TERMINAL_CODEPAGE property).

CPUCores

Gets the information about the amount of CPU cores in the system.

```
int  CPUCores() const
```

Returned value

Amount of CPU cores in the system.

Note

To get the amount of CPU cores it uses the TerminalInfoInteger() function (TERMINAL_CPU_CORES property).

MemoryPhysical

Gets the information about the physical memory (in Mb).

```
int  MemoryPhysical() const
```

Returned value

Physical memory (in Mb).

Note

To get the physical memory it uses the TerminalInfoInteger() function (TERMINAL_MEMORY_PHYSICAL property).

MemoryTotal

Gets the information about the total memory, avaliable for the client terminal/agent (in Mb).

```
int  MemoryTotal() const
```

Returned value

Total memory (in Mb), avaliable for the terminal/agent.

Note

To get the total memory it uses the TerminalInfoInteger() function (TERMINAL_MEMORY_TOTAL property).

MemoryAvailable

Gets the information about the free memory, avaliable for the client terminal/agent (in Mb).

```
int  MemoryTotal() const
```

Returned value

Free memory (in Mb), avaliable for the terminal/agent.

Note

To get the free memory it uses the TerminalInfoInteger() function (TERMINAL_MEMORY_TOTAL property).

MemoryUsed

Gets the information about the memory, used by the client terminal/agent (in Mb).

```
int  MemoryUsed() const
```

Returned value

The memory, used by the client terminal/agent (in Mb).

Note

To get the memory, used by the terminal it uses the TerminalInfoInteger() function (TERMINAL_MEMORY_USED property).

IsX64

Gets the information about the type of the client terminal.

```
bool  IsX64() const
```

Returned value

true, if 64-bit version is used, otherwise false.

Note

To get the type of the client terminal it uses the TerminalInfoInteger() function (TERMINAL_X64 property).

OpenCLSupport

Gets the information about the version of OpenCL, supported by video card.

```
int  OpenCLSupport() const
```

Returned value

The returned value has the following form: 0x00010002 = "1.2". The 0 means that OpenCL is not supported.

Note

To get the version of OpenCL it uses the TerminalInfoInteger() function (TERMINAL_OPENCL_SUPPORT property).

DiskSpace

Gets the information about free disk space, avaliable for the client terminal/agent (in Mb).

```
int  MDiskSpace() const
```

Returned value

Free disk space (in Mb), avaliable for the client terminal/agent (for files, saved to MQL5\Files folder).

Note

To get the free disk space it uses the TerminalInfoInteger() function (TERMINAL_DISK_SPACE property).

Language

Gets the information about the language in the client terminal.

```
string  Language() const
```

Returned value

Language, used in the client terminal.

Note

To get the language it uses the TerminalInfoString() function (TERMINAL_LANGUAGE property).

Name

Gets the information of the name of the client terminal.

```
string  Name() const
```

Returned value

Name of the client terminal.

Note

To get the name of the client terminal it uses the TerminalInfoString() function (TERMINAL_NAME property).

Company

Gets the information about the name of the broker.

```
string  Company() const
```

Returned value

The name of the broker.

Note

To get the broker name it uses the TerminalInfoString() function (TERMINAL_COMPANY property).

Path

Gets the client terminal folder.

```
string  Path() const
```

Returned value

The client terminal folder.

Note

To get the client terminal folder it uses the TerminalInfoString() (TERMINAL_PATH property).

DataPath

Gets the information about the terminal data folder.

```
string  DataPath() const
```

Returned value

Data folder of the client terminal.

Note

To get the client terminal data folder it uses the TerminalInfoString() function (TERMINAL_DATA_PATH property).

CommonDataPath

Gets the common data folder of all client terminals, installed on the computer.

```
string  CommonDataPath() const
```

Returned value

Common data folder.

Note

To get common data folder it uses the TerminalInfoString() function (COMMON_DATA_PATH property).

InfoInteger

Returns the value of a corresponding property of the mql5 program environment.

```
int  TerminalInfoInteger(
   int  property_id      // identifier of a property
   );
```

Parameters

property_id

[in] Identifier of a property. Can be one one of the values of the enumeration ENUM_TERMINAL_INFO_INTEGER.

Returned value

Value of int type.

Note

To get the property value it uses the TerminalInfoInteger() function.

InfoString

the function returns the value of a corresponding property of the mql5 program environment. The property must be of string type.

```
string  TerminalInfoString(
   int  property_id      // identifier of a property
   );
```

Parameters

property_id

[in] Identifier of a property. Perhaps one of the values of the enumeration ENUM_TERMINAL_INFO_STRING.

Returned value

Value of string type.

Note

To get the property value it uses the TerminalInfoString() function.

Trading Strategy Classes

This section contains technical details of working with classes for creation and testing of trading strategies and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating the trading strategies.

MQL5 Standard Library (in terms of trading strategies) is placed in the terminal directory, in the Include\Expert folder.

Base classes	Description
CExpertBase	Base class for all trading strategy classes
CExpert	Base class for Expert Advisor
CExpertSignal	Base class for Trading Signal classes
CExpertTrailing	Base class for Trailing Stop classes
CExpertMoney	Base class for Money Management classes

Trading signal classes	Description
CSignalAC	The module of signals based on market models of the indicator Accelerator Oscillator.
CSignalAMA	The module of signals based on market models of the indicator Adaptive Moving Average.
CSignalAO	The module of signals based on market models of the indicator Awesome Oscillator.
CSignalBearsPower	The module of signals based on market models of the oscillator Bears Power.
CSignalBullsPower	The module of signals based on market models of the oscillator Bulls Power.
CSignalCCI	The module of signals based on market models of the oscillator Commodity Channel Index.
CSignalDeM	The module of signals based on market models of the oscillator DeMarker.
CSignalDEMA	The module of signals based on market models of the indicator Double Exponential Moving Average.
CSignalEnvelopes	The module of signals based on market models of the indicator Envelopes.
CSignalFrAMA	The module of signals based on market models of the indicator Fractal Adaptive Moving Average.
CSignalITF	The module of filtration of signals by time.

Trading signal classes	Description
CSignalMACD	The module of signals based on market models of the oscillator MACD.
CSignalMA	The module of signals based on market models of the indicator Moving Average.
CSignalSAR	The module of signals based on market models of the indicator Parabolic SAR.
CSignalRSI	The module of signals based on market models of the oscillator Relative Strength Index.
CSignalRVI	The module of signals based on market models of the oscillator Relative Vigor Index.
CSignalStoch	The module of signals based on market models of the oscillator Stochastic.
CSignalTRIX	The module of signals based on market models of the oscillator Triple Exponential Average.
CSignalTEMA	The module of signals based on market models of the indicator Triple Exponential Moving Average.
CSignalWPR	The module of signals based on market models of the oscillator Williams Percent Range.

Trailing Stop classes	Description
CTrailingFixedPips	This class implements Trailing Stop algorithm based on fixed points
CTrailingMA	This class implements Trailing Stop algorithm based on the values of Moving Average indicator
CTrailingNone	A gag class, it doesn't uses any Trailing Stop algorithm
CTrailingPSAR	This class implements Trailing Stop algorithm based on the values of Parabolic SAR indicator

Money Management classes	Description
CMoneyFixedLot	A class with an algorithm, based on trading with predefined fixed lot size.
CMoneyFixedMargin	A class with an algorithm, based on trading with predefined fixed margin.
CMoneyFixedRisk	A class with an algorithm, based on trading with predefined risk.

CMoneyNone	A class with an algorithm, based on trading with minimal allowed lot size.
CMoneySizeOptimized	A class with an algorithm, based on trading with variable lot size, depending on the results of the previous deals.

Base classes for Expert Advisors

This section contains technical details of working with classes for creation and testing of trading strategies and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating the trading strategies.

MQL5 Standard Library (in terms of trading strategies) is placed in the terminal directory, in the Include\Expert folder.

Class	Description
CExpertBase	Base class for all trading strategy classes
CExpert	Base class for Expert Advisor
CExpertSignal	Base class for Trading Signal classes
CExpertTrailing	Base class for Trailing Stop classes
CExpertMoney	Base class for Money Management classes

CExpertBase

CExpertBase is a base class for the CExpert class and all trading strategy classes.

Description

CExpertBase provides the data and methods, which are common to all objects of the Expert Advisor.

Declaration

```
class CExpertBase : public CObject
```

Title

```
#include <Expert\CExpertBase.mqh>
```

Class Methods

Public Methods:

Initialization	
virtual Init	Class instance initialization method
virtual ValidationSettings	Checks the settings
Parameters	
Symbol	Sets the symbol
Period	Sets the timeframe
Magic	Sets the Expert Advisor ID
Indicators and Timeseries	
virtual SetPriceSeries	Sets pointers to external timeseries (price series)
virtual SetOtherSeries	Sets pointers to external timeseries (non-price series)
virtual InitIndicators	Initializes the indicators and timeseries
Access to Protected Data	
InitPhase	Gets the current phase of object initialization
TrendType	Sets trend type
UsedSeries	Gets bitmask of timeseries used
EveryTick	Sets the "Every tick" flag
Access to Timeseries	
Open	Gets the element of the Open timeseries by index

High	Gets the element of the High timeseries by index
Low	Gets the element of the Low timeseries by index
Close	Gets the element of the Close timeseries by index
Spread	Gets the element of the Spread timeseries by index
Time	Gets the element of the Time timeseries by index
TickVolume	Gets the element of the TickVolume timeseries by index
RealVolume	Gets the element of the RealVolume timeseries by index

Protected Methods:

Initialization of Timeseries	
InitOpen	Open timeseries initialization method
InitHigh	High timeseries initialization method
InitLow	Low timeseries initialization method
InitClose	Close timeseries initialization method
InitSpread	Spread timeseries initialization method
InitTime	Time timeseries initialization method
InitTickVolume	TickVolume timeseries initialization method
InitRealVolume	RealVolume timeseries initialization method
Service Methods	
virtual PriceLevelUnit	Gets the price level unit
virtual StartIndex	Gets the index of starting bar to analyze
virtual CompareMagic	Compares the Expert Advisor ID with the specified value

InitPhase

Gets the current phase of the object initialization.

```
ENUM_INIT_PHASE  InitPhase()
```

Returned value

Current phase of the object initialization.

Note

The object initialization consist of several phases:

1. Start initialization.

- start - after finish of the constructor
- finish - after successful completion of the Init(...) method.
- allowed - call of the Init(...) method
- not allowed - call of the ValidationSettings() method and other initialization methods

2. Parameters setting phase. In this phase you need to set all the object parameters, used for creation of indicators.

- start - after successful completion of the Init(...) method
- finish - after successful completion of the ValidationSettings() method
- allowed - call of Symbol(...) and Period(...) methods
- not allowed - call of the Init(...), SetPriceSeries(...), SetOtherSeries(...) and InitIndicators(...) methods

3. Checking of parameters.

- start - after successful completion of the ValidationSettings() method
- finish - after successful completion of the InitIndicators(...) method
- allowed - call of the Symbol(...), Period(...) and InitIndicators(...) methods
- not allowed - call of any other initialization methods

4. Finish of initialization.

- start - after successful completion of the InitIndicators(...) method
- not allowed - call of initialization methods

TrendType

Sets trend type.

```
void  TrendType(
   M_TYPE_TREND    value          // new value
   )
```

Parameters

value

[in] New value of trend type.

Returned value

None.

UsedSeries

Gets the bitmask of timeseries used.

```
int  UsedSeries()
```

Returned value

The list of used timeseries as bitmask.

Note

If the bit is set, the corresponding timeseries is used, if it isn't set, the timeseries is not used.

The bit-timeseries correspondence:

bit 0 - Open timeseries,
bit 1 - High timeseries,
bit 2 - Low timeseries,
bit 3 - Close timeseries,
bit 4 - Spread timeseries,
bit 5 - Time timeseries,
bit 6 - TickVolume timeseries,
bit 7 - RealVolume timeseries.

EveryTick

Sets the "Every tick" flag.

```
void  EveryTick(
   bool    value         // flag
   )
```

Parameters

value

[in] New value of a flag.

Returned value

None.

Note

If the flag is not set, the processing method is called only at new bar on the working timeframe and symbol.

Open

Gets the element of the Open timeseries by index.

```
double  Open(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Returned value

If successful, it returns the numerical value of the Open timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

High

Gets the element of the High timeseries by index.

```
double  High(
   int     ind          // index
   )
```

Parameters

ind

[in] Element index.

Returned value

If successful, it returns the numerical value of the High timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Low

Gets the element of the Low timeseries by index.

```
double  Low(
   int    ind          // index
   )
```

Parameters

ind

[in] Element index.

Returned value

If successful, it returns the numerical value of the Low timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Close

Gets the element of the Close timeseries by index.

```
double  Close(
   int    ind          // index
   )
```

Parameters

ind

[in] Element index.

Returned value

If successful, it returns the numerical value of the Close timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Spread

Gets the element of the Spread timeseries by index.

```
double  Spread(
   int    ind          // index
   )
```

Parameters

ind

[in] Element index.

Returned value

If successful, it returns the numerical value of the Spread timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Time

Gets the element of the Time timeseries by index.

```
datetime  Time(
   int    ind          // index
   )
```

Parameters

ind

[in] Element index.

Returned value

If successful, it returns the numerical value of the Time timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

TickVolume

Gets the element of the TickVolume timeseries by index.

```
long  TickVolume(
   int     ind          // Index
   )
```

Parameters

ind

[in] Element index.

Returned value

If successful, it returns the numerical value of the TickVolume timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

RealVolume

Gets the element of the RealVolume timeseries by index.

```
long  RealVolume(
   int    ind          // index
   )
```

Parameters

ind

[in] Element index.

Returned value

If successful, it returns the numerical value of the RealVolume timeseries element with specified index, otherwise it returns EMPTY_VALUE.

Note

The EMPTY_VALUE is returned in two cases:

1. Timeseries is not used (the corresponding bit is not set).
2. Element index is out of range.

Init

Initializes the object.

```
bool  Init(
   CSymbolInfo       symbol,    // symbol
   ENUM_TIMEFRAMES   period,    // timeframe
   double            point      // point
   )
```

Parameters

symbol

[in] Pointer to the object of CSymbolInfo type for access to symbol information.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

point

[in] The "weight" of 2/4-digit point.

Returned value

true if successful, otherwise false.

Symbol

Sets the symbol.

```
bool  Symbol(
   string    name     // symbol
   )
```

Parameters

name

[in] Symbol.

Returned value

true if successful, otherwise false.

Note

The setting of working symbol is necessary if the object uses the symbol, different from symbol, defined at the initialization.

Period

Sets the timeframe.

```
bool  Period(
   ENUM_TIMEFRAMES  value     // timeframe
   )
```

Parameters

value

[in] Timeframe.

Returned value

true if successful, otherwise false.

Note

The setting of working timeframe is necessary if the object uses the timeframe, different from timeframe, defined at the initialization.

Magic

Sets the Expert Advisor ID.

```
void  Magic(
   ulong  value     // magic
   )
```

Parameters

value

[in] Expert Advisor ID.

Returned value

None.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Returned value

true if successful, otherwise false.

SetPriceSeries

Sets pointers to external price series.

```
virtual bool  SetPriceSeries(
   CiOpen*    open,         // pointer
   CiHigh*    high,         // pointer
   CiLow*     low,          // pointer
   CiClose*   close         // pointer
   )
```

Parameters

open

[in] Pointer to Open timeseries.

high

[in] Pointer to High timeseries.

low

[in] Pointer to Low timeseries.

close

[in] Pointer to Close timeseries.

Returned value

true if successful, otherwise false.

Note

The setting of pointers to external timeseries (price series) is necessary if the object uses timeseries of the symbol and timeframe, different from the symbol and timeframe, defined at initialization.

SetOtherSeries

Sets pointers to external non-price series.

```
virtual bool  SetOtherSeries(
   CiSpread*       spread,        // pointer
   CiTime*         time,          // pointer
   CiTickVolume*   tick_volume,   // pointer
   CiRealVolume*   real_volume    // pointer
   )
```

Parameters

spread

[in] Pointer to Spread timeseries.

time

[in] Pointer to Time timeseries.

tick_volume

[in] Pointer to TickVolume timeseries.

real_volume

[in] Pointer to RealVolume timeseries.

Returned value

true if successful, otherwise false.

Note

The setting of pointers to external timeseries (non-price series) is necessary if the object uses timeseries of the symbol and timeframe, different from the symbol and timeframe, defined at initialization.

InitIndicators

Initializes all indicators and time series.

```
virtual bool  InitIndicators(
   CIndicators*  indicators=NULL    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The timeseries are initialized only if the object uses the symbol or timeframe, different from the symbol or timeframe, defined at initialization.

InitOpen

Initalizes the Open timeseries.

```
bool  InitOpen(
   CIndicators*  indicators    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The Open timeseries is initialized only if Expert Advisor uses the symbol/timeframe, different from the symbol/timeframe defined at initialization (and timeseries is used further).

InitHigh

Initalizes the High timeseries.

```
bool  InitHigh(
   CIndicators*  indicators    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The High timeseries is initialized only if Expert Advisor uses the symbol/timeframe, different from the symbol/timeframe defined at initialization (and timeseries is used further).

InitLow

Initalizes the Low timeseries.

```
bool  InitLow(
   CIndicators*  indicators    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The Low timeseries is initialized only if Expert Advisor uses the symbol/timeframe, different from the symbol/timeframe defined at initialization (and timeseries is used further).

InitClose

Initalizes the Close timeseries.

```
bool  InitClose(
   CIndicators*  indicators    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The Close timeseries is initialized only if Expert Advisor uses the symbol/timeframe, different from the symbol/timeframe defined at initialization (and timeseries is used further).

InitSpread

Initalizes the Spread timeseries.

```
bool  InitSpread(
   CIndicators*  indicators    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The Spread timeseries is initialized only if Expert Advisor uses the symbol/timeframe, different from the symbol/timeframe defined at initialization (and timeseries is used further).

InitTime

Initalizes the Time timeseries.

```
bool  InitTime(
   CIndicators*  indicators    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The Time timeseries is initialized only if Expert Advisor uses the symbol/timeframe, different from the symbol/timeframe defined at initialization (and timeseries is used further).

InitTickVolume

Initalizes the TickVolume timeseries.

```
bool  InitTickVolume(
   CIndicators*  indicators    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The TickVolume timeseries is initialized only if Expert Advisor uses the symbol/timeframe, different from the symbol/timeframe defined at initialization (and timeseries is used further).

InitRealVolume

Initalizes the RealVolume timeseries.

```
bool  InitRealVolume(
   CIndicators*  indicators    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The RealVolume timeseries is initialized only if Expert Advisor uses the symbol/timeframe, different from the symbol/timeframe defined at initialization (and timeseries is used further).

PriceLevelUnit

Gets the price level unit.

```
virtual double  PriceLevelUnit()
```

Returned value

The value of Price Level unit.

Note

The method of a base class returns the "weight" of the 2/4 digits point.

StartIndex

Gets the index of starting bar to analyze.

```
virtual int  StartIndex()
```

Returned value

The index of starting bar to analyze.

Note

The method returns 0 if the flag to analyze current bar is set to true (analysis from the current bar). If the flag is not set, it returns 1 (analysis from the last completed bar).

CompareMagic

Compares the Expert Advisor ID (magic) with the specified value.

```
virtual bool  CompareMagic(
   ulong  magic    // value to compare
   )
```

Parameters

magic

[in] Value to compare.

Returned value

true if they are equal, otherwise false.

CExpert

CExpert is a base class for trading strategies. It has built-in algorithms for working with time series and indicators and a set of virtual methods for trading strategy.

How to use it:

1. Prepare an algorithm of the strategy;
2. Create your own class, inherited from CExpert class;
3. Override the virtual methods in your class with your own algorithms.

Description

The CExpert class is a set of virtual methods for implementation of trading strategies.

Declaration

```
class CExpert : public CExpertBase
```

Title

```
#include <Expert\Expert.mqh>
```

Class Methods

Initialization	
Init	Class instance initialization method
virtual InitSignal	Initializes Trading Signal object
virtual InitTrailing	Initializes Trailing Stop object
virtual InitMoney	Initializes Money Management object
virtual InitTrade	Initializes Trade object
virtual ValidationSettings	Checks the settings
virtual InitIndicators	Initializes indicators and timeseries
virtual InitParameters	Parameters initialization method
virtual Deinit	Class instance deinitialization method
virtual DeinitSignal	Deinitializes Trading Signal object
virtual DeinitTrailing	Deinitializes Trailing Stop object
virtual DeinitMoney	Deinitializes Money Management object
virtual DeinitTrade	Deinitializes Trade object
virtual DeinitIndicators	Deinitializes indicators and timeseries
Parameters	
Magic	Sets the Expert Advisor ID

MaxOrders	Gets/Sets the maximum amount of allowed orders
OnTickProcess	Sets a flag to proceed the "OnTick" event
OnTradeProcess	Sets a flag to proceed the "OnTrade" event
OnTimerProcess	Sets a flag to proceed the "OnTimer" event
OnChartEventProcess	Sets a flag to proceed the "OnChartEvent" event
OnBookEventProcess	Sets a flag to proceed the "OnBookEvent" event
Event Processing Methods	
OnTick	OnTick event handler
OnTrade	OnTrade event handler
OnTimer	OnTimer event handler
OnChartEvent	OnChartEvent event handler
OnBookEvent	OnBookEvent event handler
Update Methods	
Refresh	Updates all data
Processing	
Processing	Main processing algorithm
Market Entry Methods	
CheckOpen	Checks position opening conditions
CheckOpenLong	Checks conditions to open long position
CheckOpenShort	Checks conditions to open short position
OpenLong	Opens a long position
OpenShort	Opens a short position
Market Exit Methods	
CheckClose	Checks conditions to close current position
CheckCloseLong	Checks conditions to close long position
CheckCloseShort	Checks conditions to close short position
CloseAll	Closes the opened position and deletes all orders
Close	Closes the opened position
CloseLong	Closes the long position
CloseShort	Closes the short position

Position Reverse Methods	
CheckReverse	Checks conditions to reverse opened position
CheckReverseLong	Checks conditions to reverse long position
CheckReverseShort	Checks conditions to reverse short position
ReverseLong	Performs reverse operation of long position
ReverseShort	Performs reverse operation of short position
Position/Order Trailing Methods	
CheckTrailingStop	Checks conditions to modify position parameters
CheckTrailingStopLong	Checks Trailing Stop conditions of long position
CheckTrailingStopShort	Checks Trailing Stop conditions of short position
TrailingStopLong	Performs Trailing Stop for long position
TrailingStopShort	Performs Trailing Stop for short position
CheckTrailingOrderLong	Checks Trailing Stop conditions of Buy Limit/ Stop Pending order
CheckTrailingOrderShort	Checks Trailing Stop conditions of Sell Limit/ Stop Pending order
TrailingOrderLong	Performs Trailing Stop for Buy Limit/Stop Pending order
TrailingOrderShort	Performs Trailing Stop for Sell Limit/Stop Pending order
Order Delete Methods	
CheckDeleteOrderLong	Checks conditions to delete Buy Pending order
CheckDeleteOrderShort	Checks conditions to delete Sell Pending order
DeleteOrders	Deletes all orders
DeleteOrder	Deletes Stop/Limit Pending order
DeleteOrderLong	Deletes Buy Limit/Stop pending order
DeleteOrderShort	Deletes Sell Limit/Stop pending order
Trade Volume Methods	
LotOpenLong	Gets trade volume for buy operation
LotOpenShort	Gets trade volume for sell operation
LotReverse	Gets trade volume for position reverse operation
Trade History Methods	

PrepareHistoryDate	Sets starting date for trade history tracking
HistoryPoint	Creates a checkpoint of trade history (saves number of positions, orders, deals and historical orders)
CheckTradeState	Compares the current state with the saved one and calls the corresponding event handler
Event flags	
WaitEvent	Sets the event waiting flag
NoWaitEvent	Resets the event waiting flag
Trade Event Processing Methods	
TradeEventPositionStopTake	Event handler of the "Position Stop Loss/Take Profit triggered" event
TradeEventOrderTriggered	Event handler of the "Pending Order Triggered" event
TradeEventPositionOpened	Event handler of the "Position Opened" event
TradeEventPositionVolumeChanged	Event handler of the "Position Volume Changed" event
TradeEventPositionModified	Event handler of the "Position Modified" event
TradeEventPositionClosed	Event handler of the "Position Closed" event
TradeEventOrderPlaced	Event handler of the "Pending Order Placed" event
TradeEventOrderModified	Event handler of the "Pending Order Modified" event
TradeEventOrderDeleted	Event handler of the "Pending Order Deleted" event
TradeEventNotIdentified	Event handler of the non-identified event
Service methods	
TimeframeAdd	Adds a timeframe to track
TimeframesFlags	Gets the flag indicating timeframes with a new bar

Init

Class instance initialization method.

```
bool  Init(
   string              symbol,         // symbol
   ENUM_TIMEFRAMES     period,         // timeframe
   bool                every_tick,     // flag
   ulong               magic           // magic
   )
```

Parameters

symbol

[in] Symbol.

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

every_tick

[in] Flag..

magic

[in] Expert Advisor ID (Magic number).

Returned value

true if successful, otherwise false.

Note

If every_tick is set to true, the Processing() method is called at each tick of the working symbol. otherwise, the Processing() is called only when at new bar of the working symbol.

Magic

Sets the Expert Advisor ID (magic).

```
void  Magic(
   ulong  value     // new value
   )
```

Parameters

value

[in] New value of Expert Advisor ID.

Returned value

None.

Note

It sets the value of Expert Advisor ID (magic) to the following classes: Trade, Signal, Money, Trailing.

Implementation

```
//+------------------------------------------------------------------+
//| Sets magic number for object and its dependent objects           |
//| INPUT:  value - new value of magic number.                       |
//| OUTPUT: no.                                                      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
void CExpert::Magic(ulong value)
  {
   if(m_trade!=NULL)     m_trade.SetExpertMagicNumber(value);
   if(m_signal!=NULL)    m_signal.Magic(value);
   if(m_money!=NULL)     m_money.Magic(value);
   if(m_trailing!=NULL) m_trailing.Magic(value);
//---
   CExpertBase::Magic(value);
  }
```

InitSignal

Initializes Trading Signal object.

```
virtual bool  InitSignal(
   CExpertSignal*     signal=NULL,        // pointer
   )
```

Parameters

signal

[in] Pointer to CExpertSignal class object (or its heis).

Returned value

true if successful, otherwise false.

Note

If signal is NULL, the CExpertSignal class will be used, it does nothing.

InitTrailing

Initializes Trailing Stop object.

```
virtual bool  InitTrailing(
   CExpertTrailing*     trailing=NULL,        // pointer
   )
```

Parameters

trailing

[in] Pointer to CExpertTrailing class object (or its heir).

Returned value

true if successful, otherwise false.

Note

If trailing is NULL, the ExpertTrailing class will be used, it does nothing.

InitMoney

Initializes the Money Management object.

```
virtual bool  InitMoney(
   CExpertMoney*     money=NULL,         // pointer
   )
```

Parameters

money

[in] Pointer to CExpertMoney class object (or its heir).

Returned value

true if successful, otherwise false.

Note

If money is NULL, the CExpertMoney class will be used, it uses the minimum lot.

InitTrade

Initializes the Trade object.

```
virtual bool  InitTrade(
   ulong            magic,        // magic
   CExpertTrade*    trade=NULL    // pointer
   )
```

Parameters

magic

[in] Expert Advisor ID (will be used in trade requests).

trade

[in] Pointer to CExpertTrade object.

Returned value

true if successful, otherwise false.

Deinit

Class instance deinitialization method.

```
virtual void Deinit()
```

Returned value

None.

OnTickProcess

Sets a flag to proceed the OnTick event.

```
void  OnTickOProcess(
   bool     value        // flag
   )
```

Parameters

value

[in] Flag to proceed the OnTick event.

Returned value

None.

Note

If the flag is true, the OnTick event is proceed, by default, the flag is set to true.

OnTradeProcess

Sets a flag to proceed the OnTrade event.

```
void  OnTradeProcess(
   bool     value        // flag
   )
```

Parameters

value

[in] Flag to proceed the OnTrade event.

Returned value

None.

Note

If the flag is true, the OnTrade event is proceed, by default, the flag is set to false.

OnTimerProcess

Sets a flag to proceed the OnTimer event.

```
void  OnTimerProcess(
   bool     value        // flag
   )
```

Parameters

value

[in] Flag to proceed the OnTimer event.

Returned value

None.

Note

If the flag is true, the OnTimer event is proceed, by default, the flag is set to false.

OnChartEventProcess

Sets a flag to proceed the OnChartEvent event.

```
void  OnChartEventProcess(
   bool     value        // flag
   )
```

Parameters

value

[in] Flag to proceed the OnChartEvent event.

Returned value

None.

Note

If the flag is true, the OnChartEvent event is proceed, by default, the flag is set to false.

OnBookEventProcess

Sets a flag to proceed the OnBookEvent event.

```
void  OnChartEventProcess(
   bool     value        // flag
   )
```

Parameters

value

[in] Flag to proceed the OnBookEvent event.

Returned value

None.

Note

If the flag is true, the OnBookEvent event is proceed, by default, the flag is set to false.

MaxOrders (Get Method)

Gets the maximum amount of allowed orders.

```
int  MaxOrders()
```

Returned value

Maximum amount of allowed orders.

MaxOrders (Set Method)

Sets the maximum amount of allowed orders.

```
void  MaxOrders(
   int      max_orders         // new value
   )
```

Parameters

max_orders

[in] New value of maximum amount of allowed orders.

Returned value

None.

Note

By default, the maximum amount of allowed orders = 1.

Signal

Gets the pointer to the Trade Signal object.

```
CExpertSignal*  Signal() const
```

Returned value

Pointer to the Trade Signal object.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Returned value

true if successful, otherwise false.

Note

It also checks the settings of all the Expert Advisor objects.

InitIndicators

Initializes all indicators and timeseries.

```
virtual bool  InitIndicators(
   CIndicators*  indicators=NULL    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The timeseries are initialized if the object uses the symbol or timeframe, different from the symbol or timeframe, defined at initialization. It calls consequentially InitIndicators() virtual methods of trading signal, trailing stop and money management objects.

OnTick

Event handler of the OnTick event.

```
virtual void  OnTick()
```

Returned value

None.

OnTrade

Event handler of the OnTrade event.

```
virtual void  OnTrade()
```

Returned value

None.

OnTimer

Event handler of the OnTimer event.

```
virtual void  OnTimer()
```

Returned value

None.

OnChartEvent

Event handler of the OnChartEvent event.

```
virtual void  OnChartEvent(
   const int     id,       // event id
   const long&   lparam,   // long parameter
   const double  dparam,   // double parameter
   const string  sparam    // string parameter
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type.

dparam

[in] Event parameter of double type.

sparam

[in] Event parameter of string type.

Returned value

None.

OnBookEvent

Event handler of the OnBookEvent event.

```
virtual void  OnBookEvent(
   const string&     symbol       // symbol
)
```

Parameters

symbol

[in] Symbol of OnBookEvent event.

Returned value

None.

InitParameters

Initializes parameters of Expert Advisor.

```
virtual bool  InitParameters()
```

Returned value

true if successful, otherwise false.

Note

The InitParameters() function of CExpert base class does nothing and always returns true.

DeinitTrade

Deinitializes Trade object.

```
virtual void  DeinitTrade()
```

Returned value

None.

DeinitSignal

Deinitializes Signal object.

```
virtual void  DeinitSignal()
```

Returned value

None.

DeinitTrailing

Deinitializes Trailing object.

```
virtual void  DeinitTrailing()
```

Returned value

None.

DeinitMoney

Deinitializes Money Management object.

```
virtual void  DeinitMoney()
```

Returned value

None.

DeinitIndicators

Deinitializes all indicators and time series.

```
virtual void  DeinitIndicators()
```

Returned value

None.

Note

It also deinitializes all indicators and time series of all auxiliary objects.

Refresh

Updates all data.

```
virtual bool  Refresh()
```

Returned value

true if further tick processing is needed, otherwise false.

Note

It allows to determine the need of tick processing. If it needed, it updates all quotes and time series and indicators data and returns true.

Implementation

```
//+------------------------------------------------------------------+
//| Refreshing data for processing                                   |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if successful, false otherwise.                     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::Refresh()
  {
   MqlDateTime time;
//--- refresh rates
   if(!m_symbol.RefreshRates()) return(false);
//--- check need processing
   TimeToStruct(m_symbol.Time(),time);
   if(m_period_flags!=WRONG_VALUE && m_period_flags!=0)
      if((m_period_flags & TimeframesFlags(time))==0) return(false);
   m_last_tick_time=time;
//--- refresh indicators
   m_indicators.Refresh();
//--- ok
   return(true);
  }
```

Processing

Main processing algorithm.

```
virtual bool  Processing()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It does the following steps:

1. Checks the presence of the opened position on the symbol. If there isn't opened position, skip steps №2, №3 and №4.
2. Checks conditions to reverse opened position (CheckReverse() method). If position has been "reversed", exit.
3. Checks conditions to close position (CheckClose() method). If position has been closed, skip step №4.
4. Checks conditions to modify position parameters (CheckTrailingStop() method). If position parameters has been modified, exit.
5. Check the presence of pending orders on the symbol. If there isn't any pending orders, go to step №9.
6. Checks condition to delete order (CheckDeleteOrderLong() for buy pending orders or CheckDeleteOrderShort() for sell pending orders). If the order has been deleted, go to step №9.
7. Check conditions to modify pending order parameters (CheckTrailingOrderLong() for buy orders or CheckTrailingOrderShort() for sell orders). If the order parameters has been modified, exit.
8. Exit.
9. Checks conditions to open position (CheckOpen()) method).

If you want to implement your own algorithm, you need to override the Processing() method of the heir class.

Implementation

```
//+------------------------------------------------------------------+
//| Main function                                                    |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if any trade operation processed, false otherwise.  |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::Processing()
  {
//--- check if open positions
   if(m_position.Select(m_symbol.Name()))
     {
      //--- open position is available
      //--- check the possibility of reverse the position
      if(CheckReverse()) return(true);
      //--- check the possibility of closing the position/delete pending orders
      if(!CheckClose())
        {
```

```
         //--- check the possibility of modifying the position
         if(CheckTrailingStop()) return(true);
         //--- return without operations
         return(false);
        }
     }
//--- check if placed pending orders
   int total=OrdersTotal();
   if(total!=0)
     {
      for(int i=total-1;i>=0;i--)
        {
         m_order.SelectByIndex(i);
         if(m_order.Symbol()!=m_symbol.Name()) continue;
         if(m_order.OrderType()==ORDER_TYPE_BUY_LIMIT || m_order.OrderType()==ORDER_T
           {
            //--- check the ability to delete a pending order to buy
            if(CheckDeleteOrderLong()) return(true);
            //--- check the possibility of modifying a pending order to buy
            if(CheckTrailingOrderLong()) return(true);
           }
         else
           {
            //--- check the ability to delete a pending order to sell
            if(CheckDeleteOrderShort()) return(true);
            //--- check the possibility of modifying a pending order to sell
            if(CheckTrailingOrderShort()) return(true);
           }
         //--- return without operations
         return(false);
        }
     }
//--- check the possibility of opening a position/setting pending order
   if(CheckOpen()) return(true);
//--- return without operations
   return(false);
  }
```

CheckOpen

Checks conditions to open position.

```
virtual bool  CheckOpen()
```

Returned value

true if any trade operation has been executed, otherwise false.

Note

It checks conditions to open long (CheckOpenLong()) and short (CheckOpenShort()) positions.

Implementation

```
//+------------------------------------------------------------------+
//| Check for position open or limit/stop order set                  |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckOpen()
  {
   if(CheckOpenLong())  return(true);
   if(CheckOpenShort()) return(true);
//--- return without operations
   return(false);
  }
```

CheckOpenLong

Checks conditions to open long position.

```
virtual bool  CheckOpenLong()
```

Returned value

true if any trade operation has been executed, otherwise false.

Note

It checks conditions to open long position (CheckOpenLong() method of Signal object) and opens long position (OpenLong() method) if necessary.

Implementation

```
//+------------------------------------------------------------------+
//| Check for long position open or limit/stop order set             |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckOpenLong()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- check signal for long enter operations
   if(m_signal.CheckOpenLong(price,sl,tp,expiration))
     {
      if(!m_trade.SetOrderExpiration(expiration))
        {
         m_expiration=expiration;
        }
      return(OpenLong(price,sl,tp));
     }
//--- return without operations
   return(false);
  }
```

CheckOpenShort

Checks conditions to open short position.

```
virtual bool  CheckOpenShort()
```

Returned value

true if any trade operation has been executed, otherwise false.

Note

It checks conditions to open short position (CheckOpenShort() method of Signal object) and opens a short position (OpenShort() method) if necessary.

Implementation

```
//+------------------------------------------------------------------+
//| Check for short position open or limit/stop order set            |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckOpenShort()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- check signal for short enter operations
   if(m_signal.CheckOpenShort(price,sl,tp,expiration))
     {
      if(!m_trade.SetOrderExpiration(expiration))
        {
         m_expiration=expiration;
        }
      return(OpenShort(price,sl,tp));
     }
//--- return without operations
   return(false);
  }
```

OpenLong

Opens a long position.

```
virtual bool  OpenLong(
   double    price,     // price
   double    sl,        // Stop Loss
   double    tp         // Take Profit
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Returned value

true if trade operation has been executed, otherwise false.

Note

It gets trading volume (LotOpenLong(...) method) and opens a long position (Buy() method of Trade object) if trading volume is not equal to 0.

Implementation

```
//+------------------------------------------------------------------+
//| Long position open or limit/stop order set                       |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss,                                       |
//|         tp    - take profit.                                     |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::OpenLong(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- get lot for open
   double lot=LotOpenLong(price,sl);
//--- check lot for open
   if(lot==0.0) return(false);
//---
   return(m_trade.Buy(lot,price,sl,tp));
  }
```

OpenShort

Opens a short position.

```
virtual bool  OpenShort(
   double    price,    // price
   double    sl,       // Stop Loss
   double    tp        // Take Profit
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Returned value

true if trade operation has been executed, otherwise false.

Note

It gets trading volume (LotOpenShort() method) and opens a short position (by calling Sell method of Trade object) if trading volume is not equal to 0.

Implementation

```
//+------------------------------------------------------------------+
//| Short position open or limit/stop order set                      |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss,                                       |
//|         tp    - take profit.                                     |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::OpenShort(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- get lot for open
   double lot=LotOpenShort(price,sl);
//--- check lot for open
   if(lot==0.0) return(false);
//---
   return(m_trade.Sell(lot,price,sl,tp));
  }
```

CheckReverse

Checks conditions to reverse opened position.

```
virtual bool  CheckReverse()
```

Returned value

true if any trade operation has been executed, otherwise false.

Note

It checks conditions to reverse long (CheckReverseLong()) and short (CheckReverseShort()) positions.

Implementation

```
//+------------------------------------------------------------------+
//| Check for position reverse                                       |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckReverse()
  {
   if(m_position.PositionType()==POSITION_TYPE_BUY)
     {
      //--- check the possibility of reverse the long position
      if(CheckReverseLong())  return(true);
     }
   else
      //--- check the possibility of reverse the short position
      if(CheckReverseShort()) return(true);
//--- return without operations
   return(false);
  }
```

CheckReverseLong

Checks conditions to reverse long position.

```
virtual bool  CheckReverseLong()
```

Returned value

true if any trade operation has been executed, otherwise false.

Note

It checks conditions to reverse long position (CheckReverseLong() method of Signal object) and perform reverse operation of the current long position (ReverseLong(...) method) if necessary.

Implementation

```
//+------------------------------------------------------------------+
//| Check for long position reverse                                  |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckReverseLong()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- check signal for long reverse operations
   if(m_signal.CheckReverseLong(price,sl,tp,expiration)) return(ReverseLong(price,sl,
//--- return without operations
   return(false);
  }
```

CheckReverseShort

Checks conditions to reverse short position.

```
virtual bool  CheckReverseLong()
```

Returned value

true if any trade operation has been executed, otherwise false.

Note

It checks conditions to reverse short position (CheckReverseShort() method of Signal object) and perform reverse operation of the current short position (ReverseShort() method) if necessary.

Implementation

```
//+------------------------------------------------------------------+
//| Check for short position reverse                                 |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckReverseShort()
  {
   double   price=EMPTY_VALUE;
   double   sl=0.0;
   double   tp=0.0;
   datetime expiration=TimeCurrent();
//--- check signal for short reverse operations
   if(m_signal.CheckReverseShort(price,sl,tp,expiration)) return(ReverseShort(price,s
//--- return without operations
   return(false);
  }
```

ReverseLong

Performs reverse operation of long position.

```
virtual bool  ReverseLong(
   double    price,     // price
   double    sl,        // Stop Loss
   double    tp         // Take Profit
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Returned value

true if trade operation has been executed, otherwise false.

Note

It gets the position reverse volume (LotReverse() method) and perform trade operation of the long position reverse (Sell() method of Trade object) if trading volume is not equal to 0.

Implementation

```
//+------------------------------------------------------------------+
//| Long position reverse                                            |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss,                                       |
//|         tp    - take profit.                                     |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::ReverseLong(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- get lot for reverse
   double lot=LotReverse(sl);
//--- check lot
   if(lot==0.0) return(false);
//---
   return(m_trade.Sell(lot,price,sl,tp));
  }
```

ReverseShort

Performs reverse operation of short position.

```
virtual bool  ReverseShort(
   double    price,    // price
   double    sl,       // Stop Loss
   double    tp        // Take Profit
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Returned value

true if trade operation has been executed, otherwise false.

Note

It gets position reverse volume (LotReverse(...) method) and perform trade operation of the short position reverse (Buy() method of Trade object) if trading volume is not equal to 0.

Implementation

```
//+------------------------------------------------------------------+
//| Short position reverse                                           |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss,                                       |
//|         tp    - take profit.                                     |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::ReverseShort(double price,double sl,double tp)
  {
   if(price==EMPTY_VALUE) return(false);
//--- get lot for reverse
   double lot=LotReverse(sl);
//--- check lot
   if(lot==0.0) return(false);
//---
   return(m_trade.Buy(lot,price,sl,tp));
  }
```

CheckClose

Checks conditions to close position.

```
virtual bool  CheckClose()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

1. It checks Expert Advisor Stop Out conditions (CheckClose() method of money management object). If condition is satisfied, it closes the position and deletes all orders (CloseAll(...)) and exit.
2. It checks conditions to close long or short position (CheckCloseLong() or CheckCloseShort() methods) and if position is closed, it deletes all orders (DeleteOrders() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for position close or limit/stop order delete              |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckClose()
  {
   double lot;
//--- position must be selected before call
   if((lot=m_money.CheckClose(GetPointer(m_position)))!=0.0)
      return(CloseAll(lot));
//--- check for position type
   if(m_position.PositionType()==POSITION_TYPE_BUY)
     {
      //--- check the possibility of closing the long position / delete pending order
      if(CheckCloseLong())
        {
         DeleteOrders();
         return(true);
        }
     }
   else
     {
      //--- check the possibility of closing the short position / delete pending orde
      if(CheckCloseShort())
        {
         DeleteOrders();
         return(true);
        }
     }
```

```
//--- return without operations
   return(false);
  }
```

CheckCloseLong

Checks conditions to close long position.

```
virtual bool  CheckCloseLong()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It checks conditions to close long position (CheckCloseLong() method of Signal object) and if it satisfied, it closes the opened position (CloseLong(...) method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for long position close or limit/stop order delete         |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckCloseLong()
  {
   double price=EMPTY_VALUE;
//--- check for long close operations
   if(m_signal.CheckCloseLong(price))
      return(CloseLong(price));
//--- return without operations
   return(false);
  }
```

CheckCloseShort

Checks conditions to close short position.

```
virtual bool  CheckCloseShort()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It checks conditions to close short position (CheckCloseShort() method of Signal object) and if it satisfied, it closes the position (CloseShort() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for short position close or limit/stop order delete        |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckCloseShort()
  {
   double price=EMPTY_VALUE;
//--- check for short close operations
   if(m_signal.CheckCloseShort(price))
      return(CloseShort(price));
//--- return without operations
   return(false);
  }
```

CloseAll

It performs partial of full position closing.

```
virtual bool  CloseAll(
   double    lot    // lot
   )
```

Parameters

lot

[in] Number of lots to reduce the position.

Returned value

true if trade operation has been executed, otherwise false.

Note

It performs partial of full position closing (Sell() and Buy() methods of CTrade class object for the long/short positions) and deletes all orders (DeleteOrders() method).

Implementation

```
//+------------------------------------------------------------------+
//| Position close and orders delete                                 |
//| INPUT:  lot - volume for close.                                  |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CloseAll(double lot)
  {
   bool result;
//--- check for close operations
   if(m_position.PositionType()==POSITION_TYPE_BUY) result=m_trade.Sell(lot,0,0,0);
   else                                             result=m_trade.Buy(lot,0,0,0);
   result|=DeleteOrders();
//---
   return(result);
  }
```

Close

Closes the opened position.

```
virtual bool  Close()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

Closes the position (PositionClose() method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Position close                                                   |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::Close()
  {
   return(m_trade.PositionClose(m_symbol.Name()));
  }
```

CloseLong

Closes the long position.

```
virtual bool  CloseLong(
   double    price    // price
   )
```

Parameters

price

[in] Price.

Returned value

true if trade operation has been executed, otherwise false.

Note

Closes the long position (Sell(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Long position close                                              |
//| INPUT:  price - price for close.                                 |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CloseLong(double price)
  {
   if(price==EMPTY_VALUE) return(false);
//---
   return(m_trade.Sell(m_position.Volume(),price,0,0));
  }
```

CloseShort

Closes the short position.

```
virtual bool  CloseShort(
   double    price    // price
   )
```

Parameters

price

[in] Price.

Returned value

true if trade operation has been executed, otherwise false.

Note

Closes the short position (Buy(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Short position close                                             |
//| INPUT:  price - price for close.                                 |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CloseShort(double price)
  {
   if(price==EMPTY_VALUE) return(false);
//---
   return(m_trade.Buy(m_position.Volume(),price,0,0));
  }
```

CheckTrailingStop

It checks Trailing Stop conditions of the opened position.

```
virtual bool  CheckTrailingStop()
```

Returned value

true if any trade operation has been executed, otherwise false.

Note

It checks Trailing Stop conditions of the opened position (CheckTrailingStopLong() or CheckTrailingStopShort() for long and short positions).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing stop/profit position                          |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingStop()
  {
//--- position must be selected before call
   if(m_position.PositionType()==POSITION_TYPE_BUY)
     {
      //--- check the possibility of modifying the long position
      if(CheckTrailingStopLong()) return(true);
     }
   else
     {
      //--- check the possibility of modifying the short position
      if(CheckTrailingStopShort()) return(true);
     }
//--- return without operations
   return(false);
  }
```

CheckTrailingStopLong

It checks Trailing Stop conditions of the opened long position.

```
virtual bool  CheckTrailingStopLong()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It checks Trailing Stop conditions of the opened long position (CheckTrailingStopLong(...) method of Expert Trailing object). If conditions are satisfied, it modifies the position parameters (TrailingStopLong(...) method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing stop/profit long position                     |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingStopLong()
  {
   double sl=EMPTY_VALUE;
   double tp=EMPTY_VALUE;
//--- check for long trailing stop operations
   if(m_trailing.CheckTrailingStopLong(GetPointer(m_position),sl,tp))
     {
      if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
      if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
      //--- long trailing stop operations
      return(TrailingStopLong(sl,tp));
     }
//--- return without operations
   return(false);
  }
```

CheckTrailingStopShort

It checks Trailing Stop conditions of the opened short position.

```
virtual bool  CheckTrailingStopShort()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It checks Trailing Stop conditions of the opened short position (CheckTrailingStopShort(...) method of Expert Trailing object). If conditions are satisfied, it modifies the position parameters (TrailingStopShort(...) method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing stop/profit short position                    |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingStopShort()
  {
   double sl=EMPTY_VALUE;
   double tp=EMPTY_VALUE;
//--- check for short trailing stop operations
   if(m_trailing.CheckTrailingStopShort(GetPointer(m_position),sl,tp))
     {
      if(sl==EMPTY_VALUE) sl=m_position.StopLoss();
      if(tp==EMPTY_VALUE) tp=m_position.TakeProfit();
      //--- short trailing stop operations
      return(TrailingStopShort(sl,tp));
     }
//--- return without operations
   return(false);
  }
```

TrailingStopLong

It modifies parameters of the opened long position.

```
virtual bool  TrailingStopLong(
   double    sl,    // Stop Loss
   double    tp,    // Take Profit
   )
```

Parameters

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Returned value

true if trade operation has been executed, otherwise false.

Note

The function modifies parameters of the opened long position (PositionModify(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Trailing stop/profit long position                               |
//| INPUT:  sl - new stop loss,                                      |
//|         tp - new take profit.                                    |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::TrailingStopLong(double sl,double tp)
  {
   return(m_trade.PositionModify(m_symbol.Name(),sl,tp));
  }
```

TrailingStopShort

It modifies parameters of the opened short position.

```
virtual bool  TrailingStopLong(
   double    sl,    // Stop Loss
   double    tp,    // Take Profit
   )
```

Parameters

sl

[in] Stop Loss price.

tp

[in] Take Profit price.

Returned value

true if trade operation has been executed, otherwise false.

Note

The function modifies parameters of the opened short position (PositionModify(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Trailing stop/profit short position                              |
//| INPUT:  sl - new stop loss,                                      |
//|         tp - new take profit.                                    |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::TrailingStopShort(double sl,double tp)
  {
   return(m_trade.PositionModify(m_symbol.Name(),sl,tp));
  }
```

CheckTrailingOrderLong

Checks Trailing Stop conditions of Buy Limit/Stop Pending order.

```
virtual bool  CheckTrailingOrderLong()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It checks Trailing Stop conditions for buy limit/stop pending order (CheckTrailingOrderLong() method of Trade Signals object) and modifies the order parameters if necessary (TrailingOrderLong(...) method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing long limit/stop order                         |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingOrderLong()
  {
   double price;
//--- check the possibility of modifying the long order
   if(m_signal.CheckTrailingOrderLong(GetPointer(m_order),price))
      return(TrailingOrderLong(m_order.PriceOpen()-price));
//--- return without operations
   return(false);
  }
```

CheckTrailingOrderShort

It checks Trailing Stop conditions of Sell Limit/Stop pending order.

```
virtual bool  CheckTrailingOrderShort()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It checks Trailing Stop conditions for sell limit/stop pending order (CheckTrailingOrderShort() method of Trade Signals object) and modifies the order parameters if necessary (TrailingOrderShort() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for trailing short limit/stop order                        |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckTrailingOrderShort()
  {
   double price;
//--- check the possibility of modifying the short order
   if(m_signal.CheckTrailingOrderShort(GetPointer(m_order),price))
      return(TrailingOrderShort(m_order.PriceOpen()-price));
//--- return without operations
   return(false);
  }
```

TrailingOrderLong

It modifies parameters of Buy Limit/Stop Pending order.

```
virtual bool  TrailingOrderLong(
   double    delta    // delta
   )
```

Parameters

delta

[in] Price delta.

Returned value

true if trade operation has been executed, otherwise false.

Note

It modifies parameters of Buy Limit/Stop Pending order (OrderModify(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Trailing long limit/stop order                                   |
//| INPUT:  delta - price change.                                    |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::TrailingOrderLong(double delta)
  {
   ulong  ticket=m_order.Ticket();
   double price =m_order.PriceOpen()-delta;
   double sl    =m_order.StopLoss()-delta;
   double tp    =m_order.TakeProfit()-delta;
//--- modifying the long order
   return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpir
  }
```

TrailingOrderShort

It modifies parameters of Sell Limit/Stop Pending order.

```
virtual bool  TrailingOrderShort(
   double    delta    // delta
   )
```

Parameters

delta

[in] Price delta.

Returned value

true if trade operation has been executed, otherwise false.

Note

It modifies parameters of Sell Limit/Stop Pending order (OrderModify(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Trailing short limit/stop order                                  |
//| INPUT:  delta - price change.                                    |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::TrailingOrderShort(double delta)
  {
   ulong  ticket=m_order.Ticket();
   double price =m_order.PriceOpen()-delta;
   double sl    =m_order.StopLoss()-delta;
   double tp    =m_order.TakeProfit()-delta;
//--- modifying the short order
   return(m_trade.OrderModify(ticket,price,sl,tp,m_order.TypeTime(),m_order.TimeExpir
  }
```

CheckDeleteOrderLong

It checks conditions to delete Buy Limit/Stop Pending order.

```
virtual bool  CheckDeleteOrderLong()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It checks the order expiration time. It checks conditions to delete the Sell Limit/Stop Pending order (CheckCloseLong(...) method of Signal class object) and deletes the order if condition is satisfied (DeleteOrderLong() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for delete long limit/stop order                           |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckDeleteOrderLong()
  {
   double price;
//--- check the possibility of deleting the long order
   if(m_expiration!=0 && TimeCurrent()>m_expiration)
     {
      m_expiration=0;
      return(DeleteOrderLong());
     }
   if(m_signal.CheckCloseLong(price))
      return(DeleteOrderLong());
//--- return without operations
   return(false);
  }
```

CheckDeleteOrderShort

It checks conditions to delete Sell Limit/Stop Pending order.

```
virtual bool  CheckDeleteOrderShort()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It checks the order expiration time. It checks conditions to delete the Sell Limit/Stop Pending order (CheckCloseShort(...) method of Signal class object) and deletes the order if condition is satisfied (DeleteOrderShort() method).

Implementation

```
//+------------------------------------------------------------------+
//| Check for delete short limit/stop order                          |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation processed, false otherwise.      |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::CheckDeleteOrderShort()
  {
   double price;
//--- check the possibility of deleting the short order
   if(m_expiration!=0 && TimeCurrent()>m_expiration)
     {
      m_expiration=0;
      return(DeleteOrderShort());
     }
   if(m_signal.CheckCloseShort(price))
      return(DeleteOrderShort());
//--- return without operations
   return(false);
  }
```

DeleteOrders

Deletes all orders.

```
virtual bool  DeleteOrders()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It deletes all orders (DeleteOrder() for all orders).

Implementation

```
//+------------------------------------------------------------------+
//| Delete all limit/stop orders                                     |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrders()
  {
   bool result=false;
   int  total=OrdersTotal();
//---
   for(int i=total-1;i>=0;i--)
     {
      if(m_order.Select(OrderGetTicket(i)))
        {
         if(m_order.Symbol()!=m_symbol.Name()) continue;
         result|=DeleteOrder();
        }
     }
//---
   return(result);
  }
```

DeleteOrder

Deletes the Limit/Stop Pending order.

```
virtual bool  DeleteOrder()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It deletes the Limit/Stop Pending order (OrderDelete(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Delete limit/stop order                                          |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrder()
  {
   return(m_trade.OrderDelete(m_order.Ticket()));
  }
```

DeleteOrderLong

Deletes the Buy Limit/Stop Pending order.

```
virtual bool  DeleteOrderLong()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It deletes Buy Limit/Stop Pending order (OrderDelete(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Delete long limit/stop order                                     |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrderLong()
  {
   return(m_trade.OrderDelete(m_order.Ticket()));
  }
```

DeleteOrderShort

Deletes the Sell Limit/Stop Pending order.

```
virtual bool  DeleteOrderShort()
```

Returned value

true if trade operation has been executed, otherwise false.

Note

It deletes the Sell Limit/Stop pending order (OrderDelete(...) method of CTrade class object).

Implementation

```
//+------------------------------------------------------------------+
//| Delete short limit/stop order                                    |
//| INPUT:  no.                                                      |
//| OUTPUT: true-if trade operation successful, false otherwise.     |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
bool CExpert::DeleteOrderShort()
  {
   return(m_trade.OrderDelete(m_order.Ticket()));
  }
```

LotOpenLong

Gets trade volume for buy operation.

```
double  LotOpenLong(
   double    price,    // price
   double    sl        // Stop Loss
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume (in lots) for buy operation.

Note

It gets trade volume for buy operation (CheckOpenLong(...) method of money management object).

Implementation

```
//+------------------------------------------------------------------+
//| Method of getting the lot for open long position.                |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss.                                       |
//| OUTPUT: lot for open.                                            |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
double CExpert::LotOpenLong(double price,double sl)
  {
   return(m_money.CheckOpenLong(price,sl));
  }
```

LotOpenShort

Gets trade volume for sell operation.

```
double  LotOpenShort(
   double    price,    // price
   double    sl        // Stop Loss
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume (in lots) for sell operation.

Note

It gets trade volume for sell operation (CheckOpenShort(...) method of money management object).

Implementation

```
//+------------------------------------------------------------------+
//| Method of getting the lot for open short position.               |
//| INPUT:  price - price,                                           |
//|         sl    - stop loss.                                       |
//| OUTPUT: lot for open.                                            |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
double CExpert::LotOpenShort(double price,double sl)
  {
   return(m_money.CheckOpenShort(price,sl));
  }
```

LotReverse

Gets trade volume for position reverse.

```
double  LotReverse(
   double    sl       // Stop Loss
   )
```

Parameters

sl

[in] Stop Loss price.

Returned value

Trade volume (in lots) for position reverse operation.

Note

It gets trade volume for position reverse operation (CheckReverse(...) method of money management object).

Implementation

```
//+------------------------------------------------------------------+
//| Method of getting the lot for reverse position.                  |
//| INPUT:  sl - stop loss.                                          |
//| OUTPUT: lot for open.                                            |
//| REMARK: no.                                                      |
//+------------------------------------------------------------------+
double CExpert::LotReverse(double sl)
  {
   return(m_money.CheckReverse(GetPointer(m_position),sl));
  }
```

PrepareHistoryDate

Sets starting date for tracking of trade history.

```
void  PrepareHistoryDate()
```

Note

The trade history tracking period is set from the beginning of the month (but not less than one day).

HistoryPoint

Creates a checkpoint of trade history (saves number of positions, orders, deals and historical orders).

```
void  HistoryPoint(
   bool   from_check_trade=false   // flag
   )
```

Parameters

from_check_trade=false

[in] Flag to avoid recursion.

Note

It saves the amount of positions, orders, deals and historical orders.

CheckTradeState

Compares the current state with the saved one and calls the corresponding event handler.

```
bool  CheckTradeState()
```

Returned value

true if event has been handled, otherwise - false.

Note

It checks the number of positions, orders, deals and historical orders by comparing with the values, saved by HistoryPoint() method. If trade history has changed, it calls the corresponding virtual event handler.

WaitEvent

Sets the event waiting flag.

```
void  WaitEvent(
   ENUM_TRADE_EVENTS     event          // flag
   )
```

Parameters

event

[in] Flag with events to set (ENUM_TRADE_EVENTS enumeration).

Returned value

None.

Event flags

```
//--- flags of expected events
enum ENUM_TRADE_EVENTS
  {
   TRADE_EVENT_NO_EVENT              =0,         // no expected events
   TRADE_EVENT_POSITION_OPEN         =0x1,       // flag of expecting the "opening of
   TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,       // flag of expecting of the "modific
   TRADE_EVENT_POSITION_MODIFY       =0x4,       // flag of expecting of the "modific
   TRADE_EVENT_POSITION_CLOSE        =0x8,       // flag of expecting of the "closing
   TRADE_EVENT_POSITION_STOP_TAKE    =0x10,      // flag of expecting of the "trigger
   TRADE_EVENT_ORDER_PLACE           =0x20,      // flag of expecting of the "placing
   TRADE_EVENT_ORDER_MODIFY          =0x40,      // flag of expecting of the "modific
   TRADE_EVENT_ORDER_DELETE          =0x80,      // flag of expecting of the "deletio
   TRADE_EVENT_ORDER_TRIGGER         =0x100      // flag of expecting of the "trigger
  };
```

NoWaitEvent

Resets the event waiting flag.

```
void  NoWaitEvent(
   ENUM_TRADE_EVENTS    event         // flag
   )
```

Parameters

event

[in] Flag with events to reset (ENUM_TRADE_EVENTS enumeration).

Returned value

None.

Event flags

```
//--- flags of expected events
enum ENUM_TRADE_EVENTS
  {
   TRADE_EVENT_NO_EVENT              =0,         // no expected events
   TRADE_EVENT_POSITION_OPEN         =0x1,       // flag of expecting the "opening of
   TRADE_EVENT_POSITION_VOLUME_CHANGE=0x2,       // flag of expecting of the "modific
   TRADE_EVENT_POSITION_MODIFY       =0x4,       // flag of expecting of the "modific
   TRADE_EVENT_POSITION_CLOSE        =0x8,       // flag of expecting of the "closing
   TRADE_EVENT_POSITION_STOP_TAKE    =0x10,      // flag of expecting of the "trigger
   TRADE_EVENT_ORDER_PLACE           =0x20,      // flag of expecting of the "placing
   TRADE_EVENT_ORDER_MODIFY          =0x40,      // flag of expecting of the "modific
   TRADE_EVENT_ORDER_DELETE          =0x80,      // flag of expecting of the "deletio
   TRADE_EVENT_ORDER_TRIGGER         =0x100      // flag of expecting of the "trigger
  };
```

TradeEventPositionStopTake

Event handler of the "Position Stop Loss/Take Profit triggered" event.

```
virtual bool  TradeEventPositionStopTake()
```

Returned value

The CExpert class method does nothing and always returns true.

TradeEventOrderTriggered

Event handler of the "Pending Order triggered" event.

```
virtual bool  TradeEventOrderTriggered()
```

Returned value

The CExpert class method does nothing and always returns true.

TradeEventPositionOpened

Event handler of the "Position opened" event.

```
virtual bool  TradeEventPositionOpened()
```

Returned value

The CExpert class method does nothing and always returns true.

TradeEventPositionVolumeChanged

Event handler of the "Position volume changed" event.

```
virtual bool  TradeEventPositionVolumeChanged()
```

Returned value

The CExpert class method does nothing and always returns true.

TradeEventPositionModified

Event handler of the "Position modified" event.

```
virtual bool  TradeEventPositionModified()
```

Returned value

The CExpert class method does nothing and always returns true.

TradeEventPositionClosed

Event handler of the "Position closed" event.

```
virtual bool  TradeEventPositionClosed()
```

Returned value

The CExpert class method does nothing and always returns true.

TradeEventOrderPlaced

Event handler of the "Pending order placed" event.

```
virtual bool  TradeEventOrderPlaced()
```

Returned value

The CExpert class method does nothing and always returns true.

TradeEventOrderModified

Event handler of the "Pending order modified" event.

```
virtual bool  TradeEventOrderModified()
```

Returned value

The CExpert class method does nothing and always returns true.

TradeEventOrderDeleted

Event handler of the "Pending order deleted" event.

```
virtual bool  TradeEventOrderDeleted()
```

Returned value

The CExpert class method does nothing and always returns true.

TradeEventNotIdentified

Event handler of the non-identified event.

```
virtual bool  TradeEventNotIdentified()
```

Returned value

The CExpert class method does nothing and always returns true.

Note

Note that several trade events can arrive, in such cases it's difficult to identify them.

TimeframeAdd

Add a timeframe for tracking.

```
void  TimeframeAdd(
   ENUM_TIMEFRAMES    period         // timeframe
   )
```

Parameters

period

[in] Timeframe (ENUM_TIMEFRAMES enumeration).

Returned value

None.

TimeframesFlags

The method returns the flag indicating the timeframes with a new bar.

```
int  TimeframesFlags(
   MqlDateTime&    time         // variable for time
   )
```

Parameters

time

[in] Variable of MqlDateTime type for new time, passed by reference.

Returned value

It returns the flag, that indicates timeframes with a new bar.

CExpertSignal

CExpertSignal is a base class for trading signals, it does nothing (except CheckReverseLong() and CheckReverseShort() methods) but provides the interfaces.

How to use it:

1. Prepare an algorithm for trading signals;
2. Create your own trading signal class, inherited from CExpertSignal class;
3. Override the virtual methods in your class with your own algorithms.

You can find an examples of trading signal classes in the Expert\Signal\ folder.

Description

CExpertSignal is a base class for implementation of trading signal algorithms.

Declaration

```
class CExpertSignal : public CExpertBase
```

Title

```
#include <Expert\ExpertSignal.mqh>
```

Class Methods

Initialization	
virtual InitIndicators	Initializes indicators and time series
virtual ValidationSettings	Checks the settings
virtual AddFilter	Adds a filter to combined signal
Access to Protected Data	
BasePrice	Sets base price level
UsedSeries	Gets the flags of timeseries used
Parameters Setting	
Weight	Sets the value of "Weight" parameter
PatternsUsage	Sets the value of "PatternsUsage" parameter
General	Sets the value of "General" parameter
Ignore	Sets the value of "Ignore" parameter
Invert	Sets the value of "Invert" parameter
ThresholdOpen	Sets the value of "ThresholdOpen" parameter
ThresholdClose	Sets the value of "ThresholdClose" parameter

PriceLevel	Sets the value of "PriceLevel" parameter
StopLevel	Sets the value of "StopLevel" parameter
TakeLevel	Sets the value of "TakeLevel" parameter
Expiration	Sets the value of "Expiration" parameter
Magic	Sets the value of "Magic" parameter
Checking Trading Conditions	
virtual CheckOpenLong	Checks conditions to open long position
virtual CheckCloseLong	Checks conditions to close long position
virtual CheckOpenShort	Checks conditions to open short position
virtual CheckCloseShort	Checks conditions to close short position
virtual CheckReverseLong	Checks conditions of long position reversal
virtual CheckReverseShort	Checks conditions of short position reversal
Trade Parameters Setting	
virtual OpenLongParams	Sets parameters for long position opening
virtual OpenShortParams	Sets parameters for short position opening
virtual CloseLongParams	Sets parameters for long position closing
virtual CloseShortParams	Sets parameters for short position closing
Checking of Order Trailing Conditions	
virtual CheckTrailingOrderLong	Checks conditions to modify parameters of Buy Pending order
virtual CheckTrailingOrderShort	Checks conditions to modify parameters of Sell Pending order
Methods to Check Formation of Market Orders	
virtual LongCondition	Gets the result of checking of buy conditions
virtual ShortCondition	Gets the result of checking of sell conditions
virtual Direction	Gets the "weighted" direction of price

BasePrice

Sets base price level.

```
void  BasePrice(
   double    value          // new value
   )
```

Parameters

value

[in] New value of Base price level.

Returned value

None.

UsedSeries

Gets the flags of the timeseries used.

```
int  BasePrice()
```

Returned value

Flags of the used timeseries (if the symbol/timeframe corresponds to the working symbol/timeframe), otherwise 0.

Weight

Sets new value of "Weight" parameter.

```
void  Weight(
   double    value          // new value
   )
```

Parameters

value

[in] New value of "Weight".

Returned value

None.

PatternUsage

Sets new value of "PatternsUsage" parameter.

```
void  PatternUsage(
   double    value          // new value
   )
```

Parameters

value

[in] New value of "PatternsUsage".

Returned value

None.

General

Sets new value of "General" parameter.

```
void  General(
   int     value          // new value
   )
```

Parameters

value

[in] New value of "General".

Returned value

None.

Ignore

Sets new value of "Ignore" parameter.

```
void  Ignore(
   long     value          // new value
   )
```

Parameters

value

[in] New value of "Ignore".

Returned value

None.

Invert

Sets new value of "Invert" parameter.

```
void  Invert(
   long     value          // new value
   )
```

Parameters

value

[in] New value of "Invert".

Returned value

None.

ThresholdOpen

Sets new value of "ThresholdOpen" parameter.

```
void  ThresholdOpen(
   long     value          // new value
   )
```

Parameters

value

[in] New value of "ThresholdOpen".

Returned value

None.

Note

The range of "ThresholdOpen" parameter is from 0 to 100. Used when "voting" to open position.

ThresholdClose

Sets new value of "ThresholdClose" parameter.

```
void  ThresholdOpen(
   long     value          // new value
   )
```

Parameters

value

[in] New value of "ThresholdClose".

Returned value

None.

Note

The range of "ThresholdClose" parameter is from 0 to 100. Used when "voting" to close position.

PriceLevel

Sets new value of "PriceLevel" parameter.

```
void  PriceLevel(
   double    value          // new value
   )
```

Parameters

value

[in] New value of "PriceLevel".

Returned value

None.

Note

The value of "PriceLevel" is defined in price level units. The numerical values of price level unit is returned by PriceLevelUnit() method. The "PriceLevel" is used to define the open price relative to the base price.

StopLevel

Sets new value of "StopLevel" parameter.

```
void  StopLevel(
   double    value         // new value
   )
```

Parameters

value

[in] New value of "StopLevel".

Returned value

None.

Note

The value of "StopLevel" is defined in price level units. The numerical values of price level unit is returned by PriceLevelUnit() method. The "StopLevel" is used to define the Stop Loss price relative to the open price.

TakeLevel

Sets new value of "TakeLevel" parameter.

```
void  TakeLevel(
   double    value          // new value
   )
```

Parameters

value

[in] New value of "TakeLevel".

Returned value

None.

Note

The value of "TakeLevel" is defined in price level units. The numerical values of price level unit is returned by PriceLevelUnit() method. The "TakeLevel" is used to define the Take Profit price relative to the open price.

Expiration

Sets the value of "Expiration" parameter.

```
void  Expiration(
   int     value          // new value
   )
```

Parameters

value

[in] New value of "Expiration".

Returned value

None.

Note

The value of "Expiration" parameter is defined in bars. It used as Expiration time for Pending Orders (when trading using pending orders).

Magic

Sets the value of "Magic" parameter.

```
void  Magic(
   int     value          // new value
   )
```

Parameters

value

[in] New value of "Magic" (Expert Advisor ID).

Returned value

None.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Returned value

true if successful, otherwise false.

InitIndicators

Initializes all indicators and time series.

```
virtual bool  InitIndicators(
   CIndicators*  indicators    // pointer
   )
```

Parameters

indicators

[in] Pointer to collection of indicators and timeseries.

Returned value

true if successful, otherwise false.

Note

The timeseries are initialized only if the object uses the symbol or timeframe, different from the symbol or timeframe, defined at initialization.

AddFilter

Adds a filter to the composite signal.

```
virtual bool  InitIndicators(
   CExpertSignal*  filter    // pointer
   )
```

Parameters

indicators

[in] Pointer to filter object.

Returned value

true if successful, otherwise false.

CheckOpenLong

Checks conditions to open long position.

```
virtual bool  CheckOpenLong(
   double&    price,          // price
   double&    sl,             // Stop Loss
   double&    tp,             // Take Profit
   datetime&  expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference.

Returned value

true if condition is satisfied, otherwise false.

CheckOpenShort

Checks conditions to open short position.

```
virtual bool  CheckOpenShort(
   double&    price,          // price
   double&    sl,             // Stop Loss
   double&    tp,             // Take Profit
   datetime&  expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference.

Returned value

true if condition is satisfied, otherwise false.

OpenLongParams

Sets parameters to open long position.

```
virtual bool  OpenLongParams(
   double&    price,          // price
   double&    sl,             // Stop Loss
   double&    tp,             // Take Profit
   datetime&  expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference.

Returned value

true if successful, otherwise false.

OpenShortParams

Sets parameters to open short position.

```
virtual bool  OpenShortParams(
   double&    price,          // price
   double&    sl,             // Stop Loss
   double&    tp,             // Take Profit
   datetime&  expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference.

Returned value

true if successful, otherwise false.

CheckCloseLong

Checks conditions to close long position.

```
virtual bool  CheckCloseLong(
   double&  price      // price
   )
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Returned value

true if condition is satisfied, otherwise false.

CheckCloseShort

Checks conditions to close short position.

```
virtual bool  CheckCloseShort(
   double&  price      // price
   )
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Returned value

true if condition is satisfied, otherwise false.

CloseLongParams

Sets parameters to close long position.

```
virtual bool  CloseLongParams(
   double&    price          // price
   )
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Returned value

true if successful, otherwise false.

CloseShortParams

Sets parameters to close short position.

```
virtual bool  CloseShortParams(
   double&    price          // price
   )
```

Parameters

price

[in][out] Variable for close price, passed by reference.

Returned value

true if successful, otherwise false.

CheckReverseLong

Checks conditions of long position reversal.

```
virtual bool  CheckReverseLong(
   double&    price,          // price
   double&    sl,             // Stop Loss
   double&    tp,             // Take Profit
   datetime&  expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference.

Returned value

true if condition is satisfied, otherwise false.

CheckReverseShort

Checks conditions of short position reversal.

```
virtual bool  CheckReverseShort(
   double&    price,          // price
   double&    sl,             // Stop Loss
   double&    tp,             // Take Profit
   datetime&  expiration      // expiration
   )
```

Parameters

price

[in][out] Variable for reversal price, passed by reference.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

expiration

[in][out] Variable for expiration time, passed by reference.

Returned value

true if condition is satisfied, otherwise false.

CheckTrailingOrderLong

Checks conditions to modify parameters of Buy Pending order.

```
virtual bool  CheckTrailingOrderLong(
   COrderInfo*     order,          // order
   double&         price           // price
   )
```

Parameters

order

[in] Pointer to COrderInfo class object.

price

[in][out] Variable for Stop Loss price.

Returned value

true if condition is satisfied, otherwise false.

CheckTrailingOrderShort

Checks conditions to modify parameters of Sell Pending order.

```
virtual bool  CheckTrailingOrderShort(
   COrderInfo*     order,          // order
   double&         price           // price
   )
```

Parameters

order

[in] Pointer to COrderInfo class object.

price

[in][out] Variable for Stop Loss price.

Returned value

true if condition is satisfied, otherwise false.

LongCondition

Checks conditions to open long position.

```
virtual int  LongCondition()
```

Returned value

Then conditions are satisfied, it returns the value from 1 to 100 (depending on "strength" of a signal), if there isn't a signal to open long position, it returns 0.

Note

The LongCondition() method of a base class has no implementation of checking of conditions to open long position and always returns 0.

ShortCondition

Checks conditions to open short position.

```
virtual int  ShortCondition()
```

Returned value

Then conditions are satisfied, it returns the value from 1 to 100 (depending on "strength" of a signal), if there isn't a signal to open short position, it returns 0.

Note

The ShortCondition() method of a base class has no implementation of checking of conditions to open short position and always returns 0.

Direction

Returns the value of "weighted" direction.

```
virtual double  Direction()
```

Returned value

It returns the value>0 when upward direction (probably) and the returned value<0 when downward direction. The absolute value depends on the "strength" of a signal.

Note

If the filters are used, the result will depend on the filters.

CExpertTrailing

CExpertTrailing is a base class for trailing algorithms, it does nothing but provides the interfaces.

How to use it:

1. Prepare an algorithm for trailing;
2. Create your own trailing class, inherited from CExpertTrailing class;
3. Override the virtual methods in your class with your own algorithms.

You can find an examples of trailing classes in the Expert\Trailing\ folder.

Description

CExpertTrailing is a base class for implementation of trailing stop algoritms.

Declaration

```
class CExpertTrailing : public CExpertBase
```

Title

```
#include <Expert\ExpertTrailing.mqh>
```

Class Methods

Checking of Trailing Stop Conditions	
virtual CheckTrailingStopLong	Checks conditions to modify parameters of the long position
virtual CheckTrailingStopShort	Checks conditions to modify parameters of the short position

CheckTrailingStopLong

Checks conditions to modify parameters of the long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,     // pointer
   double&         sl,           // Stop Loss
   double&         tp            // Take Profit
   )
```

Parameters

position

[in] Pointer to CPositionInfo class object.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

Returned value

true if condition is satisfied, otherwise false.

Note

The CheckTrailingStopLong(...) method of the base class always returns false.

CheckTrailingStopShort

Checks conditions to modify parameters of the short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,     // pointer
   double&         sl,           // Stop Loss
   double&         tp            // Take Profit
   )
```

Parameters

position

[in] Pointer to CPositionInfo class object.

sl

[in][out] Variable for Stop Loss price, passed by reference.

tp

[in][out] Variable for Take Profit price, passed by reference.

Returned value

true if condition is satisfied, otherwise false.

Note

The CheckTrailingStopShort(...) method of the base class always returns false.

CExpertMoney

CExpertMoney is a base class for money and risk management algorithms.

Description

CExpertMoney is a base class for implementation of money and risk management classes.

Declaration

```
class CExpertMoney : public CObject
```

Title

```
#include <Expert\ExpertMoney.mqh>
```

Class Methods

Access to Protected Data	
Percent	Sets the value of "Risk percent" parameter
Initialization	
virtual ValidationSettings	Checks the settings
Checking of Trading Conditions	
virtual CheckOpenLong	Gets the volume for long position
virtual CheckOpenShort	Gets the volume for short position
virtual CheckReverse	Gets the volume for reverse of the position
virtual CheckClose	Checks conditions to close the opened position

Percent

Sets the value of "Risk percent" parameter.

```
void  Percent(
   double  percent      // risk percent
   )
```

Parameters

percent

[in] Risk percent.

Returned value

None.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Returned value

true if successful, otherwise false.

Note

The ValidationSettings() method of the base class always returns true.

CheckOpenLong

Gets the volume for long position.

```
virtual double  CheckOpenLong(
   double  price,     // price
   double  sl         // Stop Loss
   )
```

Parameters

price

[in] Opening price of long position.

sl

[in] Stop Loss price.

Returned value

Trade volume for long position.

CheckOpenShort

Gets the volume for short position.

```
virtual double  CheckOpenShort(
   double  price,     // price
   double  sl         // Stop Loss
   )
```

Parameters

price

[in] Opening price for short position.

sl

[in] Stop Loss price.

Returned value

Trade volume for short position.

CheckReverse

Gets the volume for reverse of the position.

```
virtual double  CheckReverse(
   CPositionInfo*  position,     // pointer
   double          sl            // Stop Loss
   )
```

Parameters

position

[in] Pointer to CPositionInfo class object.

sl

[in] Stop Loss price.

Returned value

Volume for reverse of the position.

CheckClose

Checks conditions to close the opened position.

```
virtual double  CheckClose()
```

Returned value

true if condition is satisfied, otherwise false.

Modules of Trade Signals

The standard delivery of the client terminal includes a set of ready-made modules of trade signals for "MQL5 Wizard". When creating an Expert Advisor in MQL5 Wizard, you can use any combination of the modules of trade signals (up to 64). The final decision on a trade operation is made on the basis of complex analysis of signals obtained from all included modules. The detailed description of the mechanism of making trade decisions is given below.

The standard delivery includes the following modules of signals:

- Signals of the Indicator Accelerator Oscillator
- Signals of the Indicator Adaptive Moving Average
- Signals of the Indicator Awesome Oscillator
- Signals of the Oscillator Bears Power
- Signals of the Oscillator Bulls Power
- Signals of the Oscillator Commodity Channel Index
- Signals of the Oscillator DeMarker
- Signals of the Indicator Double Exponential Moving Average
- Signals of the Indicator Envelopes
- Signals of the Indicator Fractal Adaptive Moving Average
- Signals of the Intraday Time Filter
- Signals of the Oscillator MACD
- Signals of the Indicator Moving Average
- Signals of the Indicator Parabolic SAR
- Signals of the Oscillator Relative Strength Index
- Signals of the Oscillator Relative Vigor Index
- Signals of the Oscillator Stochastic
- Signals of the Oscillator Triple Exponential Average
- Signals of the Indicator Triple Exponential Moving Average
- Signals of the Oscillator Williams Percent Range

The Mechanism of Making Trade Decisions on the Basis of Signal Modules

The mechanism of making trade decisions can be represented as the following list of basic principles:

- Each of the modules of signals has its set of market modules (certain combination of prices and values of an indicator).
- Each market model has a significance that may vary with the range of 1 to 100. The higher is the significance, the stronger the model is.
- Each of the models generates a forecast of direction of the price movement.
- A forecast of a module is the result of search for embedded models, and it is outputted as a number within the range of -100 to 100. The sign determines the direction of forecast movement (negative sign means the price will fall, positive sign means the price will rise). The absolute value

corresponds to the strength of the best found model.

- The forecast of each module is sent to the final "voting" with a weight coefficient of 0 to 1 specified in its settings ("Weight").
- The result of voting is a number within the range of -100 to 100, where the sign determines direction of the forecast movement, and the absolute value characterizes the strength of the signal. It is calculated as the arithmetical mean of weighted forecasts of all the modules of signals.

Each generated Expert Advisor has two adjustable settings — threshold levels of opening and closing a position (ThresholdOpen and ThresholdClose) that can be equal to a value in the range of 0 to 100. If the strength of final signal exceeds a threshold level, a trade operation that corresponds to the sign of the signal is performed.

Examples

Consider an Expert Advisor with the following threshold levels: ThresholdOpen=20 and ThresholdClose=90. Two modules of signals participate in making decisions on trade operations: the MA module with weight 0.4 and the Stochastic module with weight 0.8. Let's analyze two variants of obtained trade signals:

Variant 1.

The price crossed the rising MA upwards. This case corresponds to one of the market models implemented in the MA module. This model implies a rise of price. Its significance is equal to 100. At the same time, the Stochastic oscillator turned down and formed a divergence with price. This case corresponds to one of the models implemented in the Stochastic module. This model implies a fall of price. The weight of this model is 80.

Let's calculate the result of final "voting". The rate obtained from the MA module is calculated as 0.4 * 100 = 40. The value from the Stochastic module is calculated as 0.8 * (-80) = -64. The final value value is calculated as the arithmetical mean of these two rates: (40 - 64)/2 = -12. The result of voting is the signal for selling with relative strength equal to 12. The threshold level that is equal to 20 is not reached. Thus a trade operation is not performed.

Variant 2.

The price crossed the rising MA downwards. This case corresponds to one of the models implemented in the MA module.This model implies a rise of price. Its significance is equal to 10. At the same time, the Stochastic oscillator turned down and formed a divergence with price. This case corresponds to one of the models implemented in the Stochastic module. This model implies a fall of price. The weight of this model is 80.

Let's calculate the result of final "voting". The rate obtained from the MA module is calculated as 0.4 * 10 = 4. The value from the Stochastic module is calculated as 0.8 * (-80) = -64. The final value is calculated as the arithmetical mean of these two rates: (4 - 64)/2 = -30. The result of voting is the signal for selling with relative strength equal to 30. The threshold level that is equal to 20 is reached. Thus the result is the signal for opening a short position.

a) Divergence of the price and the Stochastic oscillator (variants 1 and 2).

b) The price crossed the MA indicator upwards (variant 1).

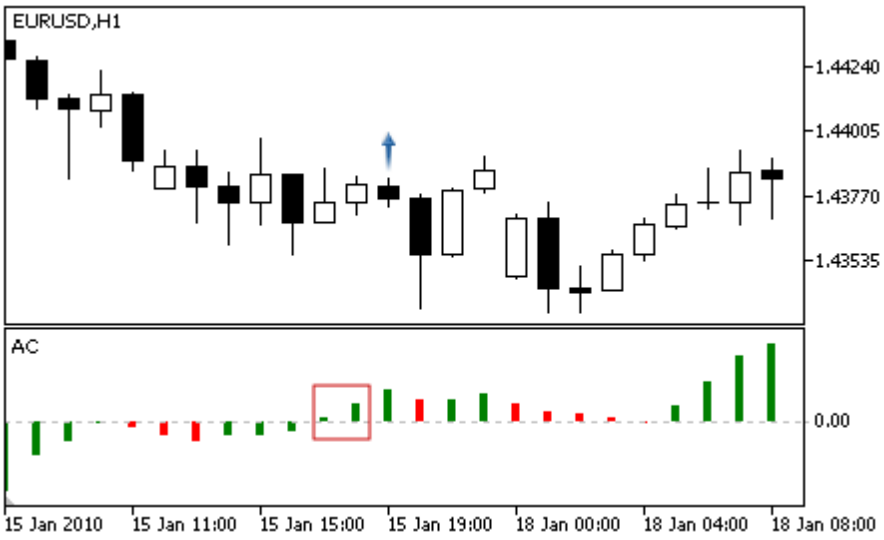
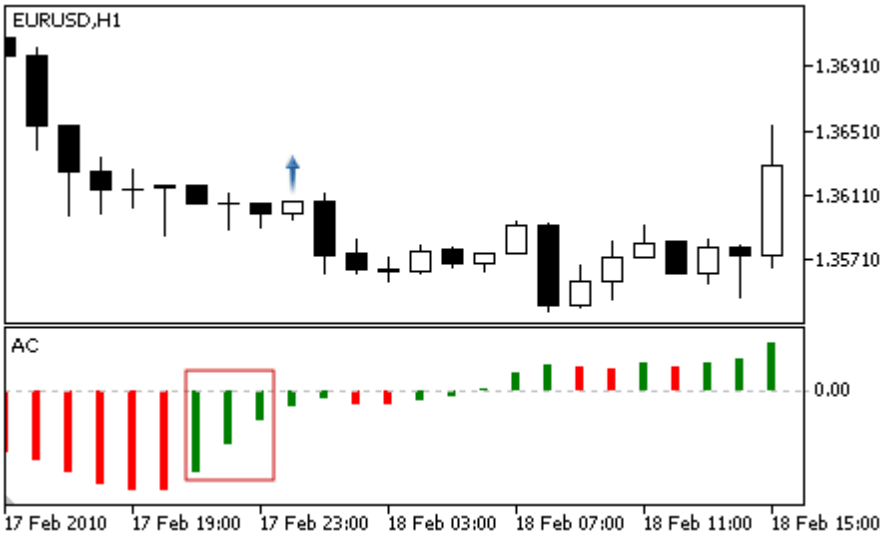
c) The price crossed the MA indicator downwards (variants 2).

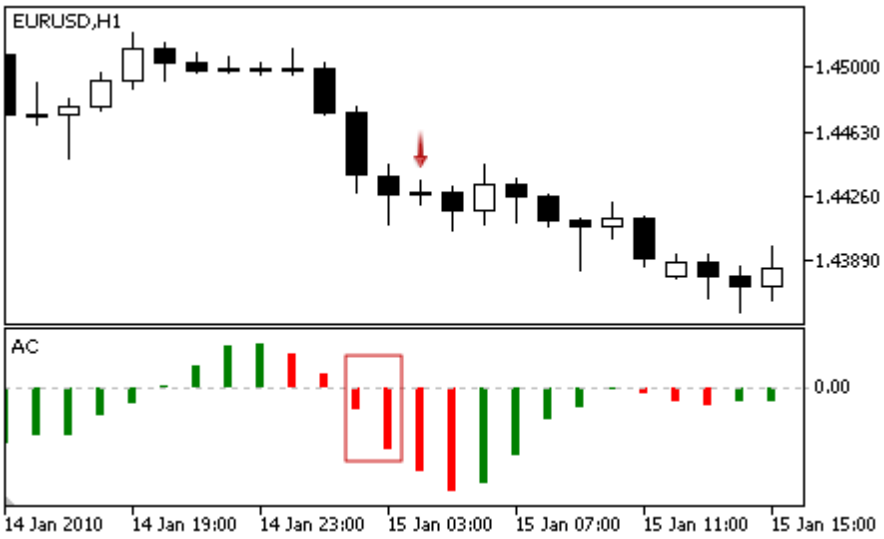
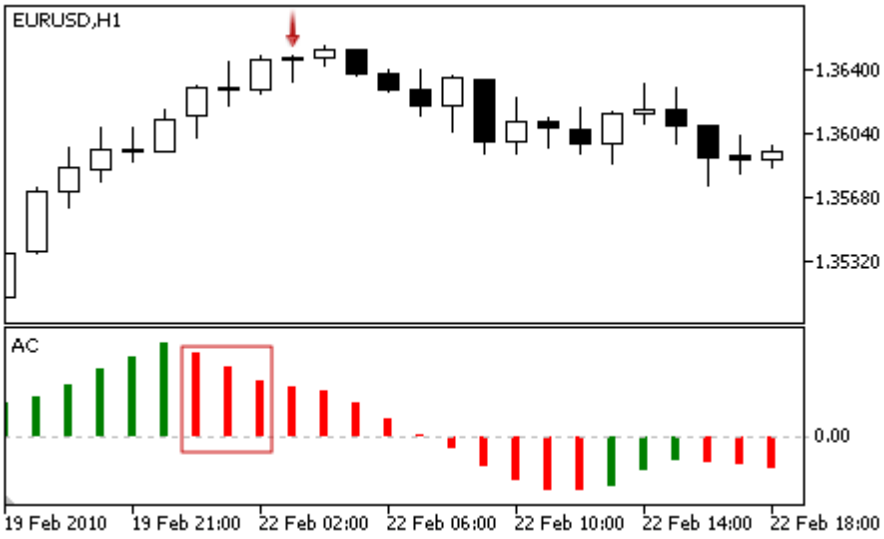
Signals of the Indicator Accelerator Oscillator

This module is based on the market models of the indicator Accelerator Oscillator. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• The indicator value is above 0 and it rises at the analyzed and at the previous bars.  • The indicator value is below 0 and it rises at the analyzed and at the previous bars. 
For selling	• The indicator value is below 0 and it falls at the analyzed and at the previous bars.

Signal Type	Description of Conditions
	 • The indicator value is below 0 and it falls at the analyzed and at the previous bars. 
No objections to buying	The indicator value grows at the analyzed bar.
No objections to selling	The indicator value falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

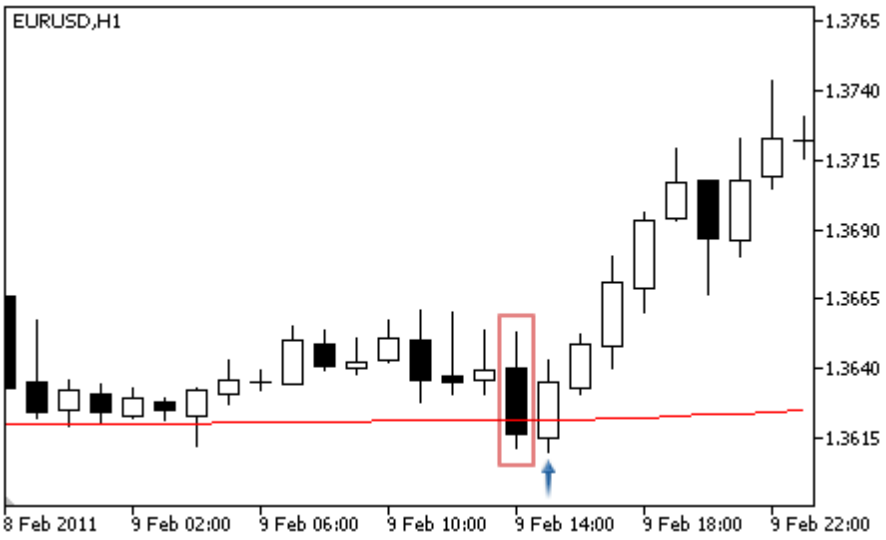
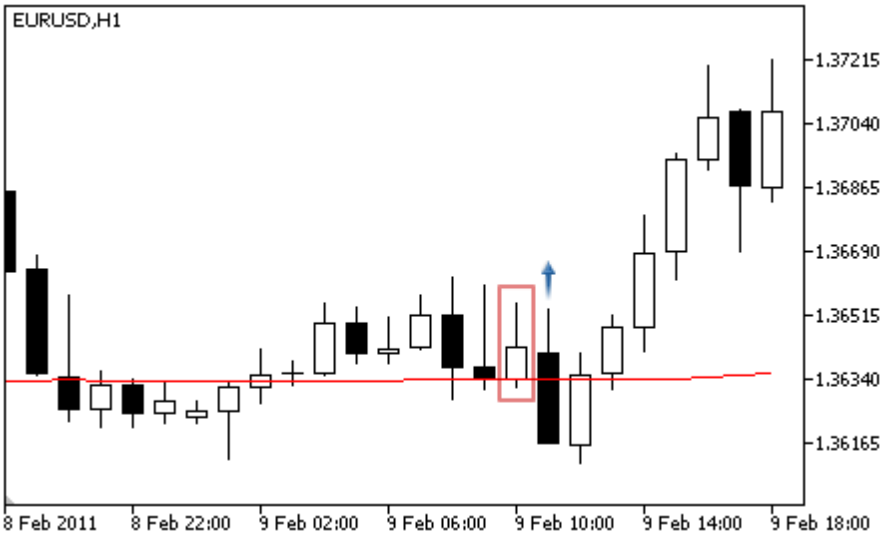
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.

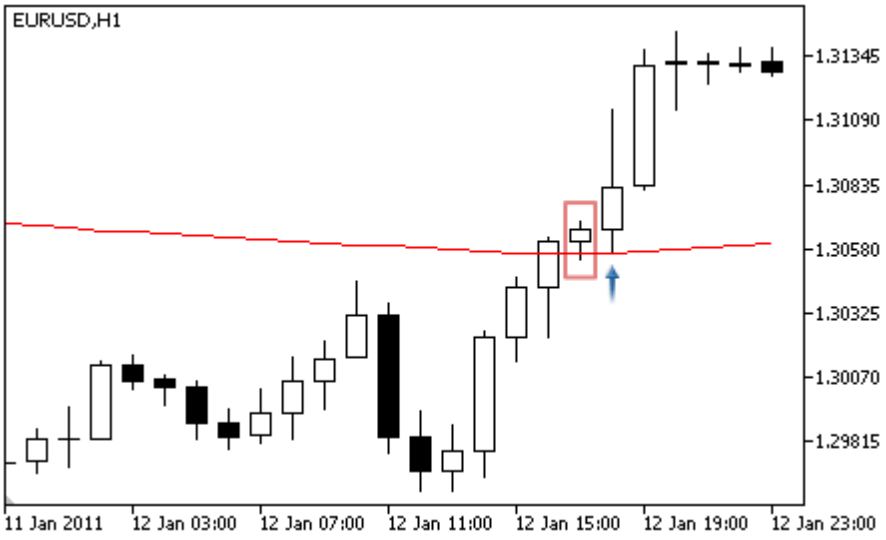
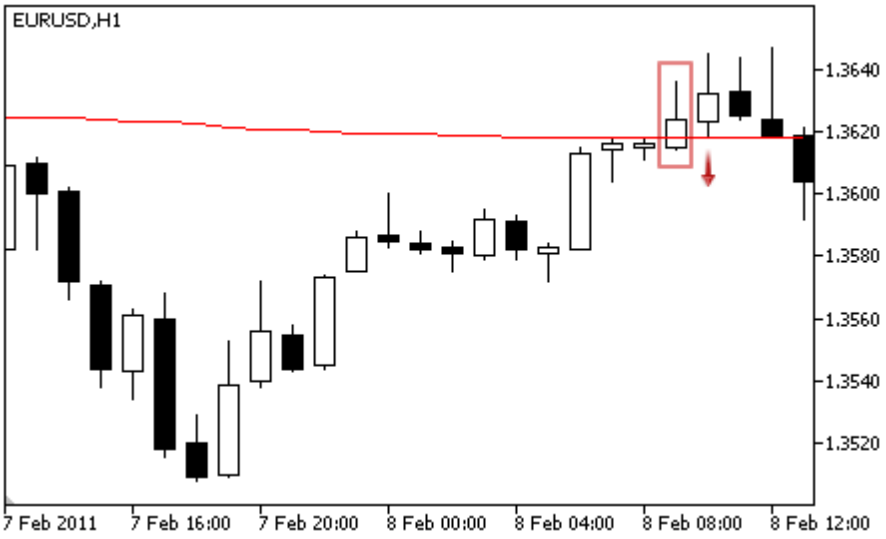
Signals of the Indicator Adaptive Moving Average

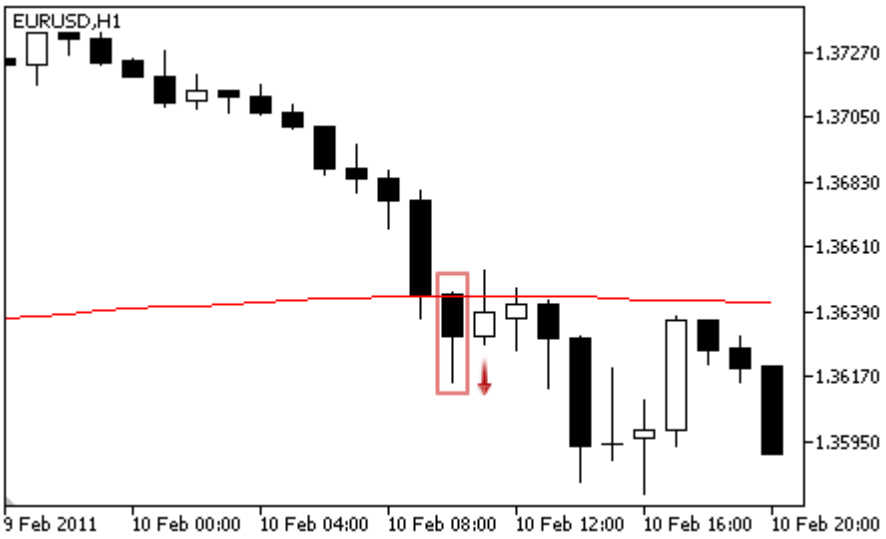
This module is based on the market models of the indicator Adaptive Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

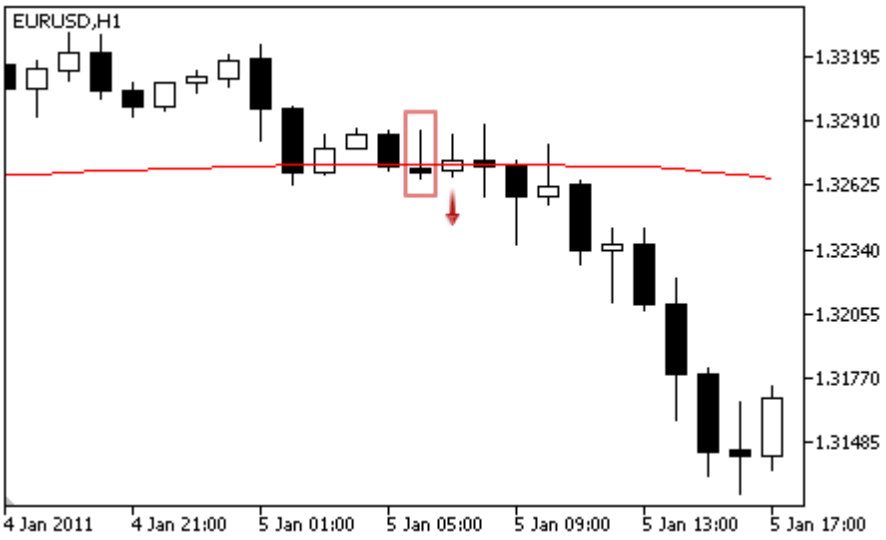
Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal).  • **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	• The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is above the indicator, and the Low price is below the indicator) and the indicator rises (weak signal). 
For selling	• The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal).  • **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td> 

The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is below the indicator, and the High price is above the indicator) and the indicator falls (weak signal).

</td></tr>
<tr><td>No objections to buying</td><td>The prices is above the indicator.</td></tr>
<tr><td>No objections to selling</td><td>The price is below the indicator.</td></tr>
</table>

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

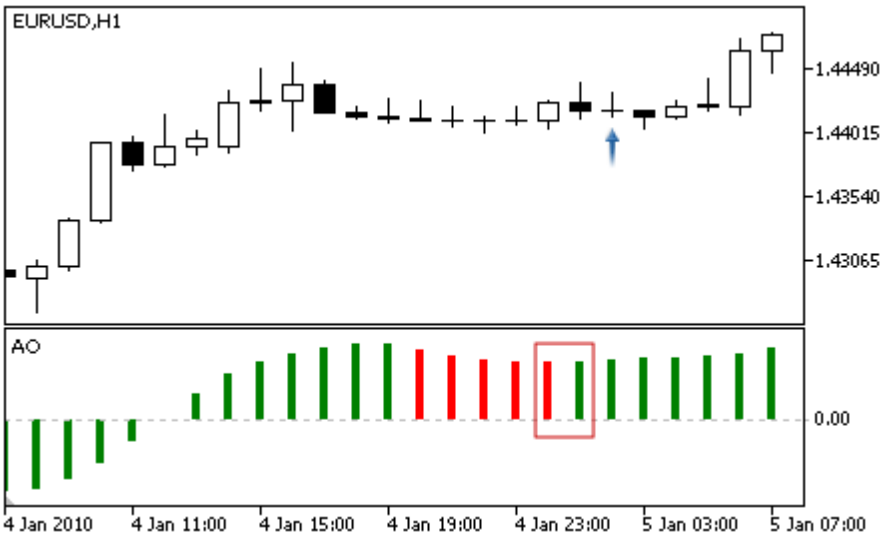
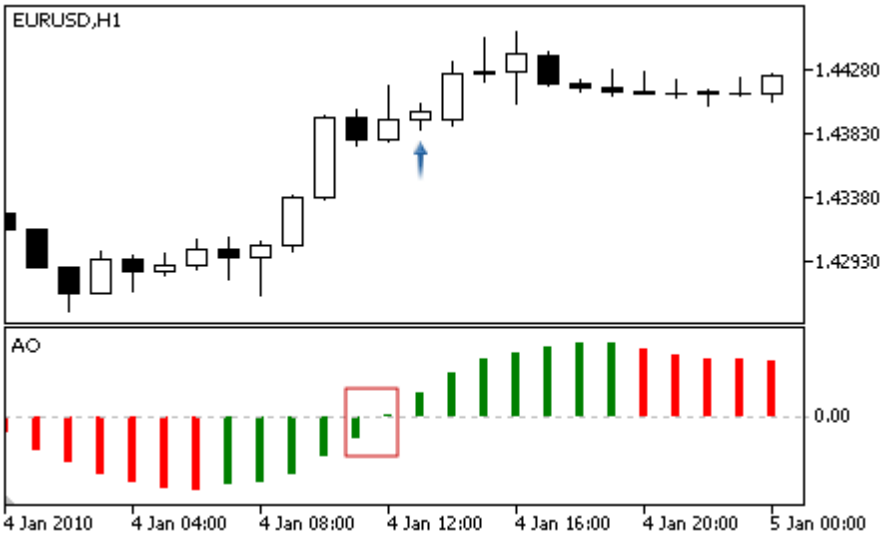
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shit of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

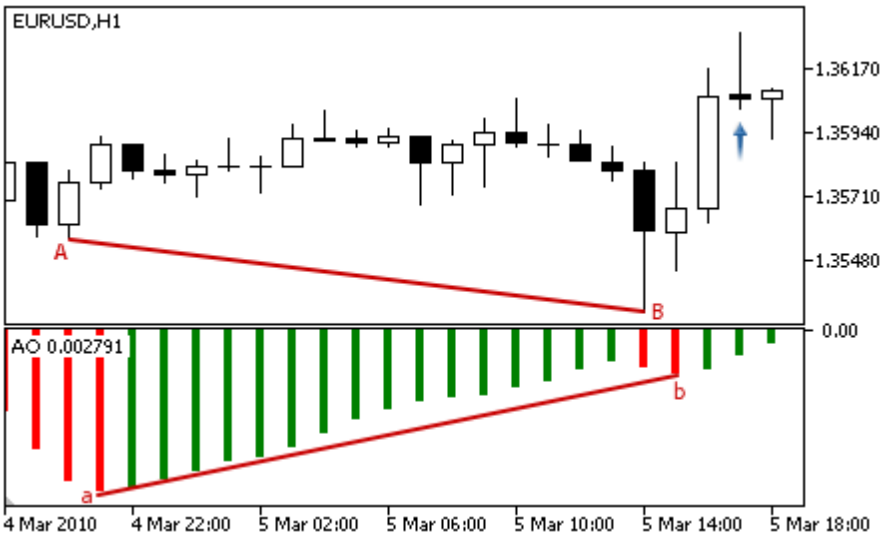
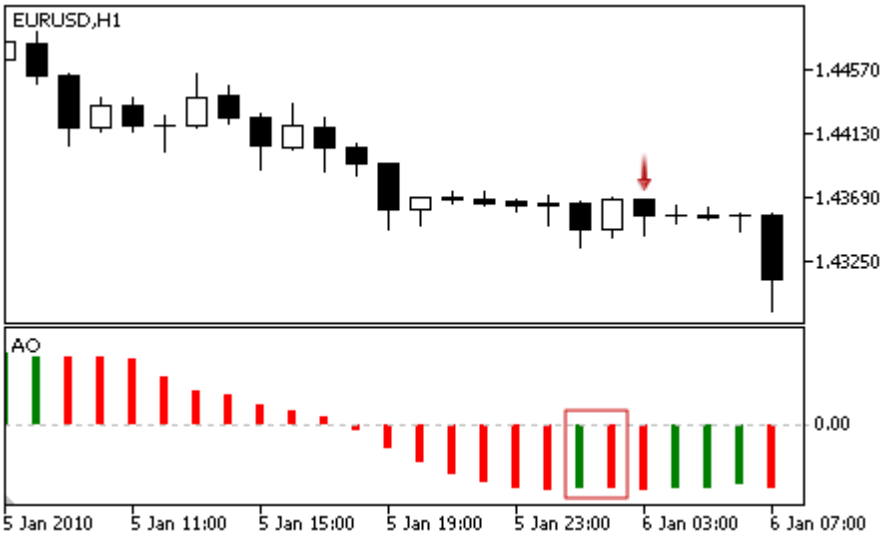
Signals of the Indicator Awesome Oscillator

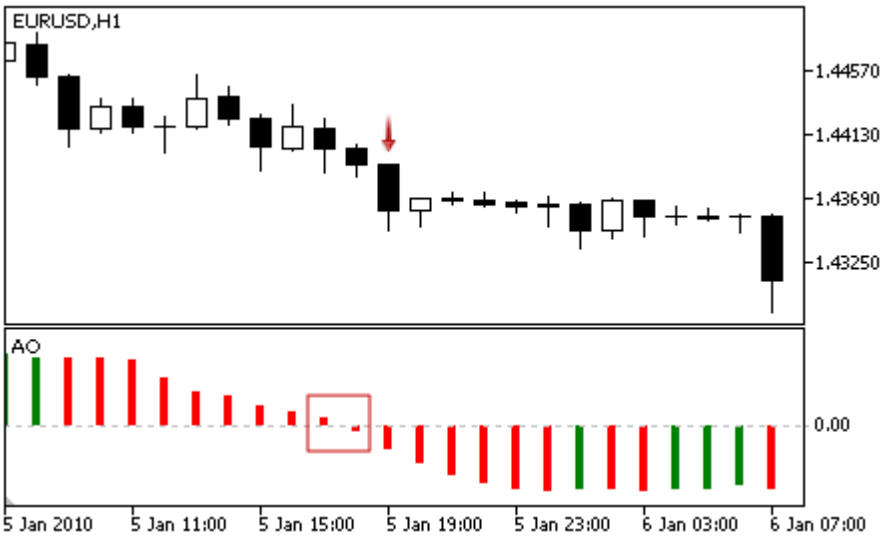
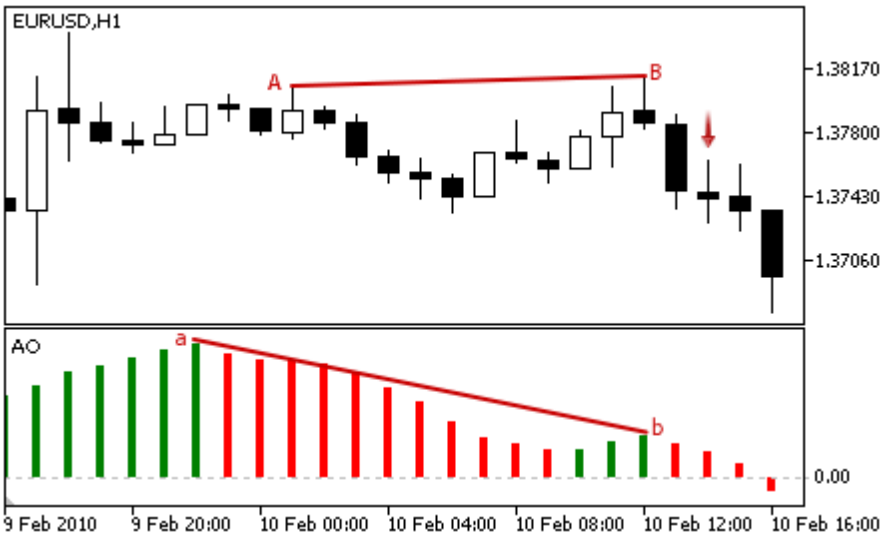
This module of signals is based on the market models of the indicator Awesome Oscillator. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• **Saucer** — value of the indicator at the analyzed bar rises, and it fell at the previous bars; at that, both values are above 0.  • **Crossing the zero line** — value of the indicator is above 0 at the analyzed bar, and it is below 0 at the previous bar.  • **Divergence** — the first analyzed bottom of the indicator is shallower than the

Signal Type	Description of Conditions
	previous one, and the corresponding price valley is deeper than the previous one. In addition, the indicator must not rise above the zero level. 
For selling	• **Saucer** — value of the indicator at the analyzed bar falls, and it rose at the previous bars; at that, both values are below 0.  • **Crossing the zero line** — value of the indicator is below 0 at the analyzed bar, and it is above 0 at the previous bar.

Signal Type	Description of Conditions
	 • **Divergence** — the first analyzed peak of the indicator is lower than the previous one, and the corresponding price peak is higher than the previous one. In addition, the indicator must not falls below the zero level. 
No objections to buying	The indicator value grows at the analyzed bar.
No objections to selling	The indicator value falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

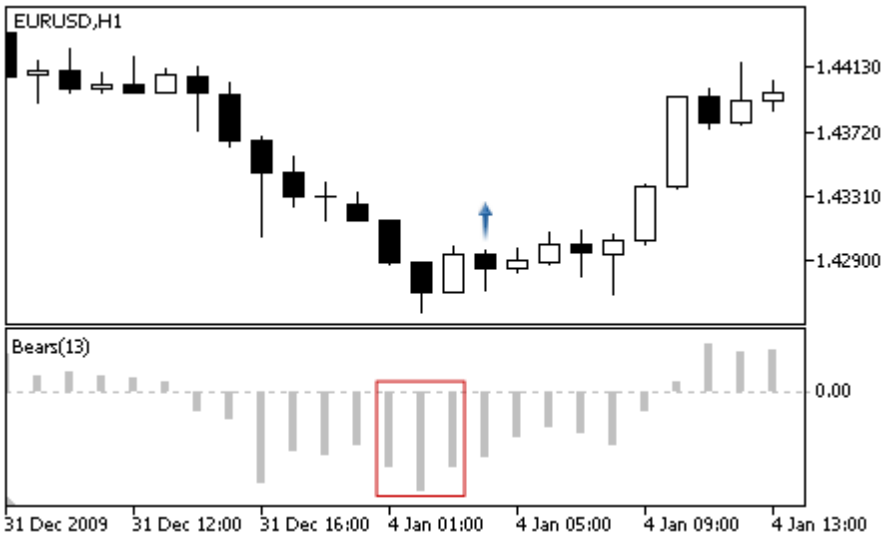
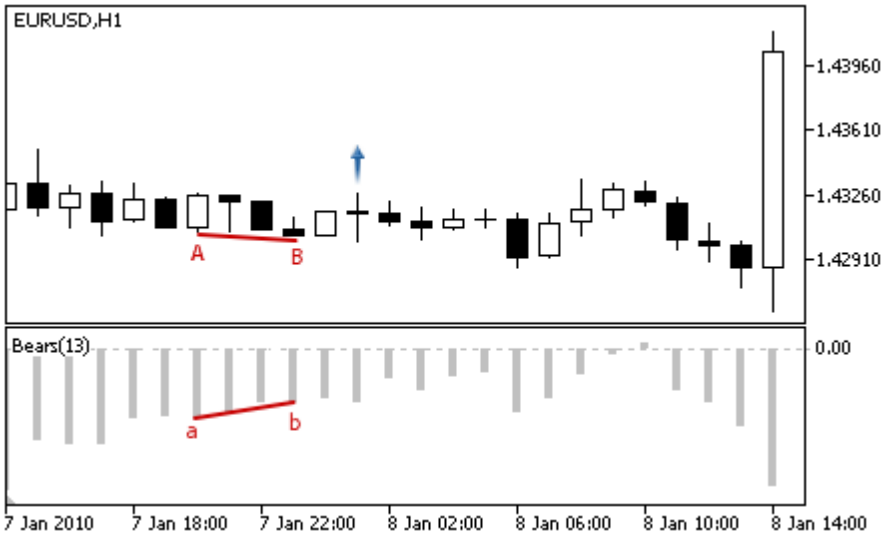
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.

Signals of the Oscillator Bears Power

This module of signals is based on the market models of the oscillator Bears Power. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• **Reverse** — the oscillator turned upwards and its value at the analyzed bar is below 0.  • **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. In addition, the oscillator must not rise above the zero level. 
For selling	No signals for selling.

Signal Type	Description of Conditions
No objections to buying	Value of the oscillator is less than 0.
No objections to selling	No signals.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

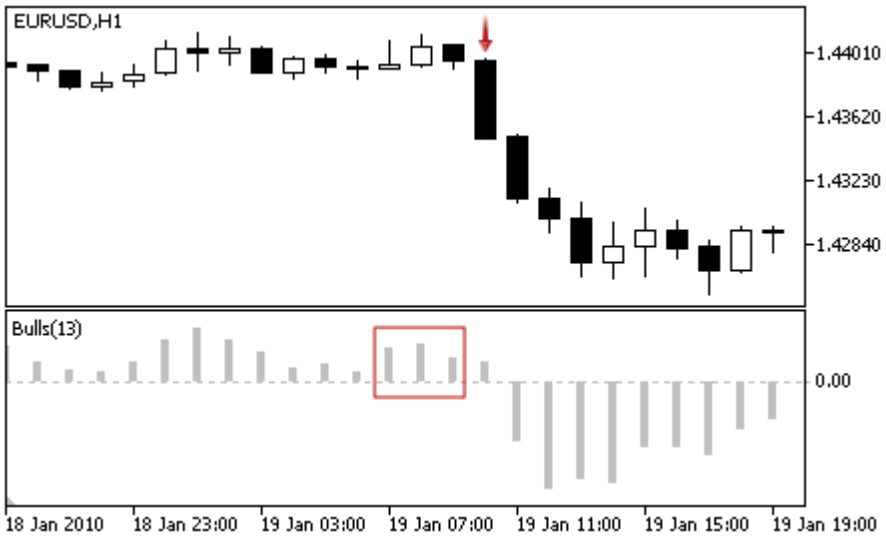
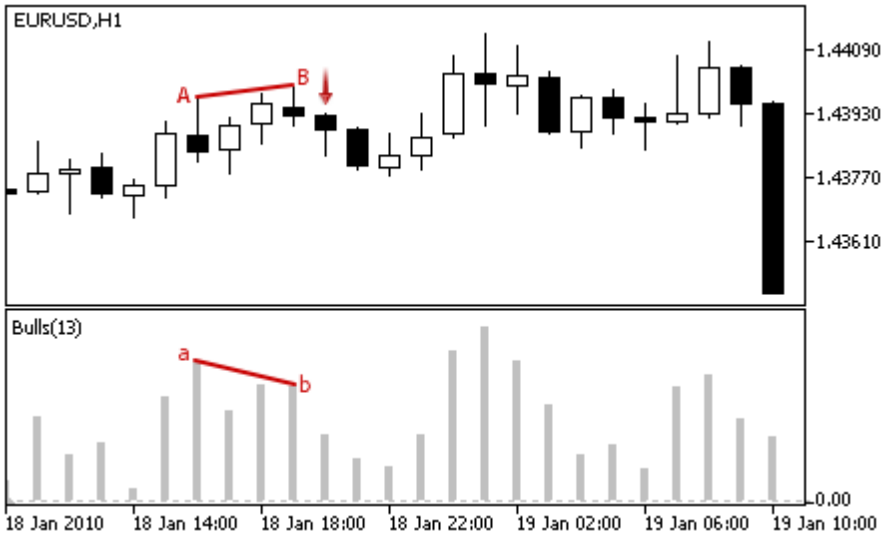
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodBears	Period of calculation of the oscillator.

Signals of the Oscillator Bulls Power

This module of signals is based on the market models of the oscillator Bulls Power. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	No signals for buying.
For selling	• **Reverse** — the oscillator turned downwards and its value at the analyzed bar is above 0.  • **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak. In addition, the oscillator must not fall below the zero level. 

Signal Type	Description of Conditions
No objections to buying	No signals.
No objections to selling	Value of the oscillator is greater than 0.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

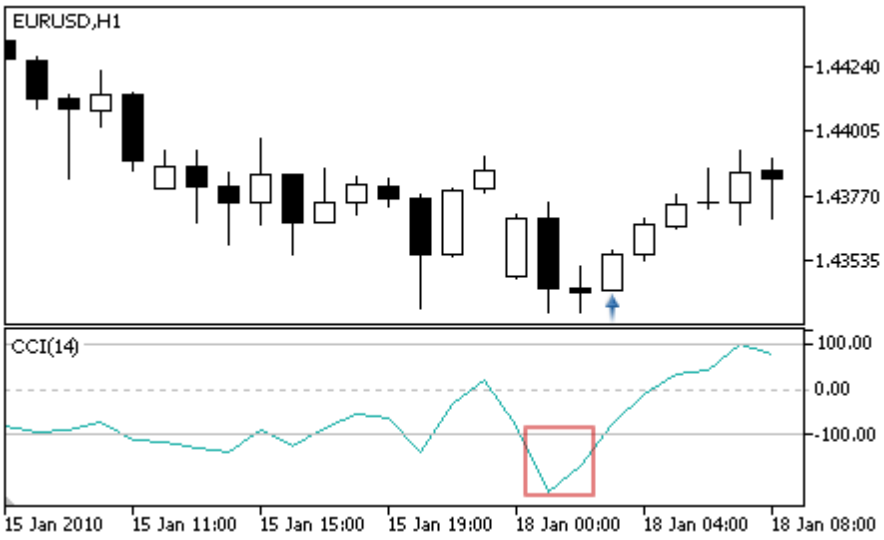
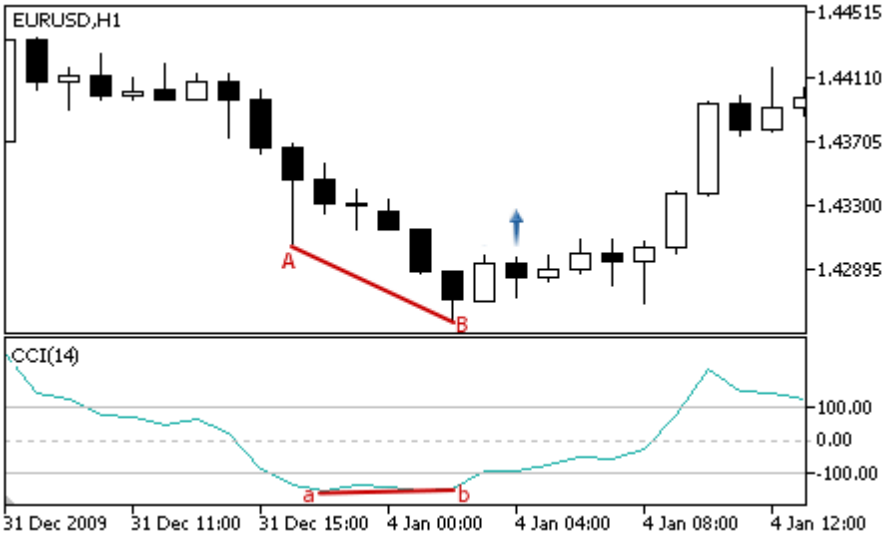
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodBulls	Period of calculation of the oscillator.

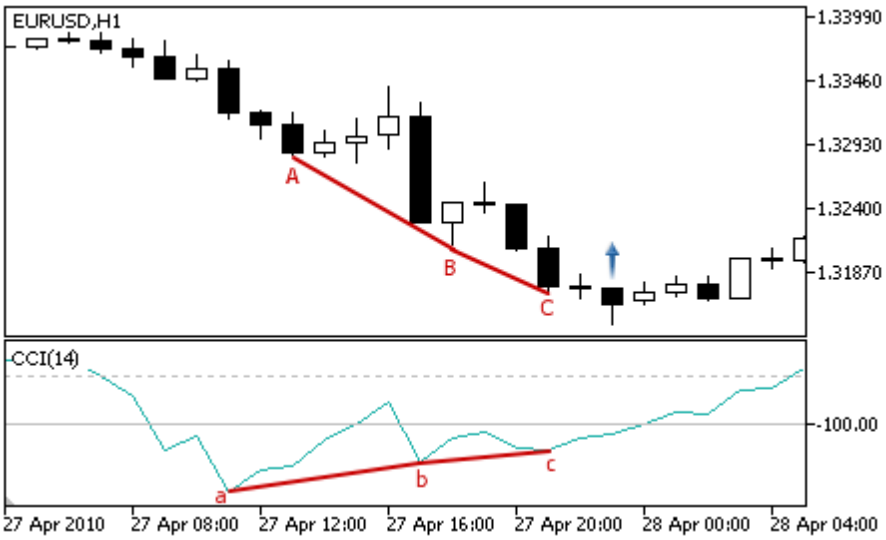
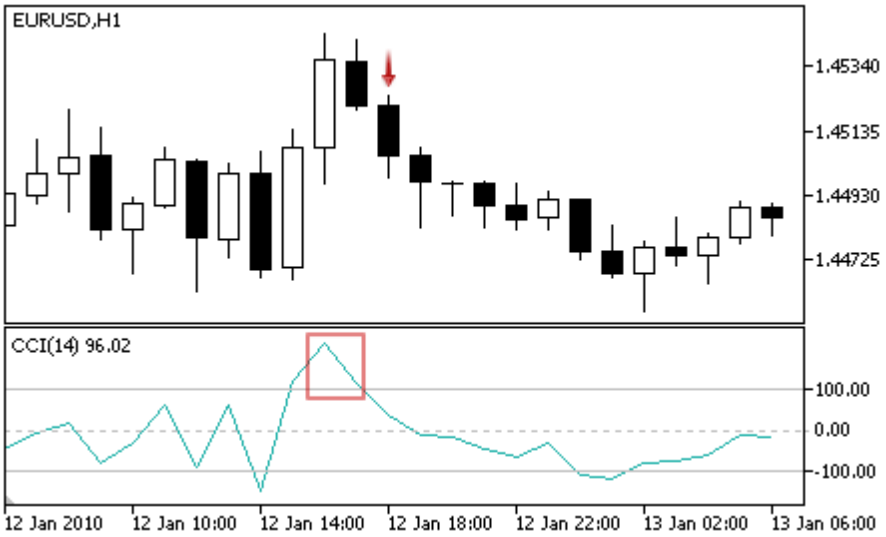
Signals of the Oscillator Commodity Channel Index

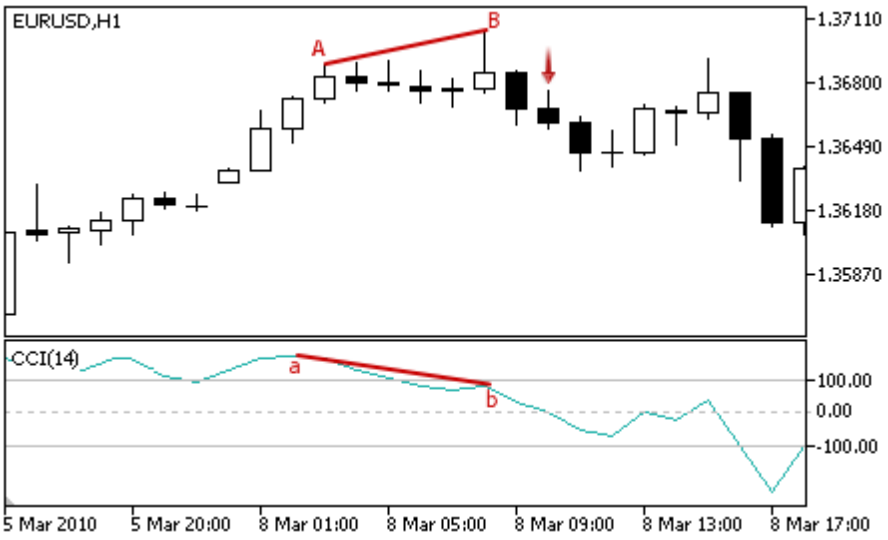
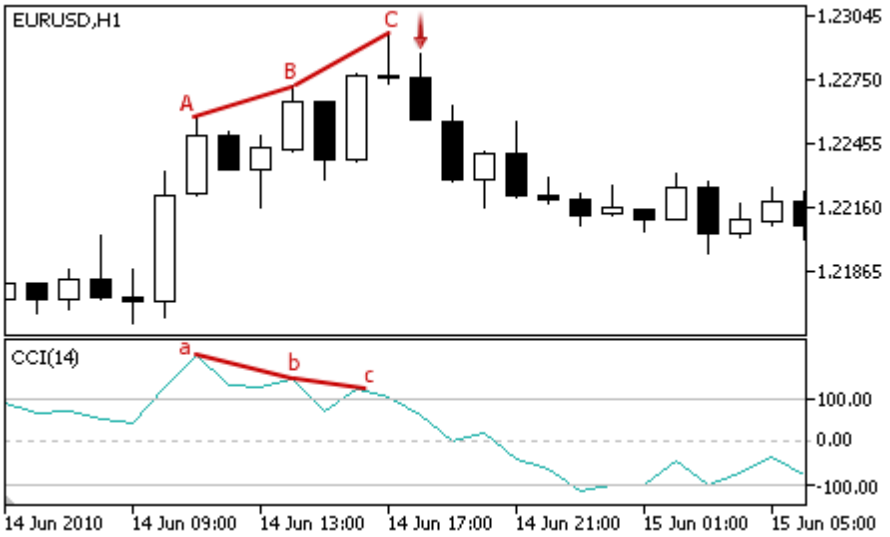
This module of signals is based on the market models of the oscillator Commodity Channel Index. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• **Reverse behind the level of overselling** — the oscillator turned upwards and its value at the analyzed bar is behind the level of overselling (default value is -100).  • **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.  • **Double divergence** — the oscillator form three consequent bottoms, each of them

Signal Type	Description of Conditions
	is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one. 
For selling	• **Reverse behind the overbought level** — the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is 100).  • **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

Signal Type	Description of Conditions
	 • **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

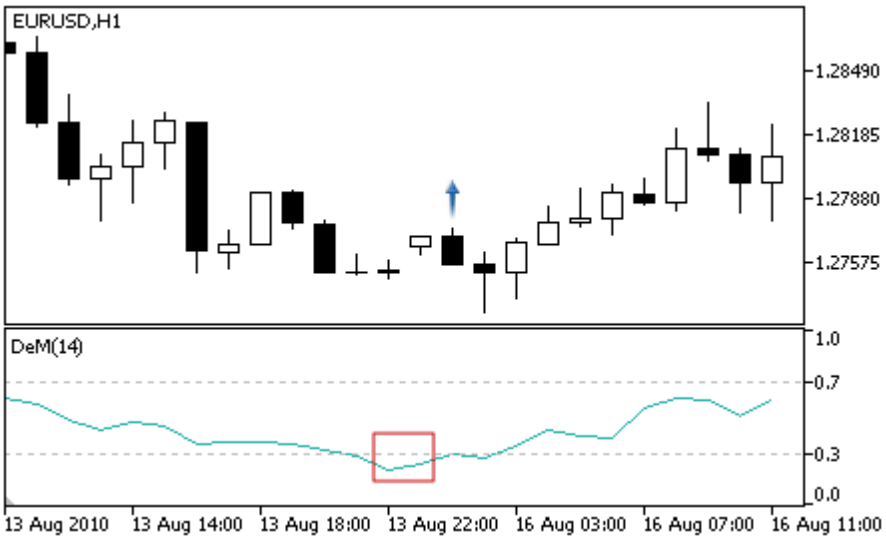
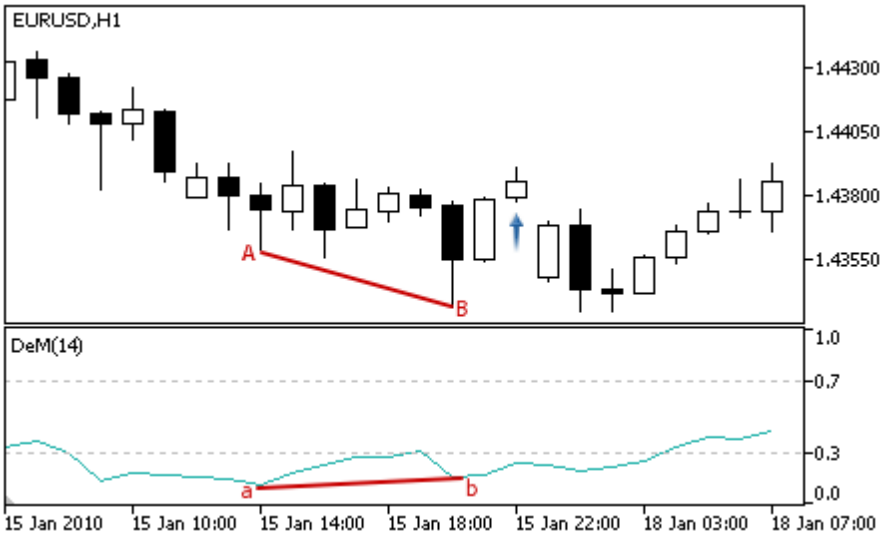
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodCCI	Period of calculation of the oscillator.
Applied	A price series used for calculation of the oscillator.

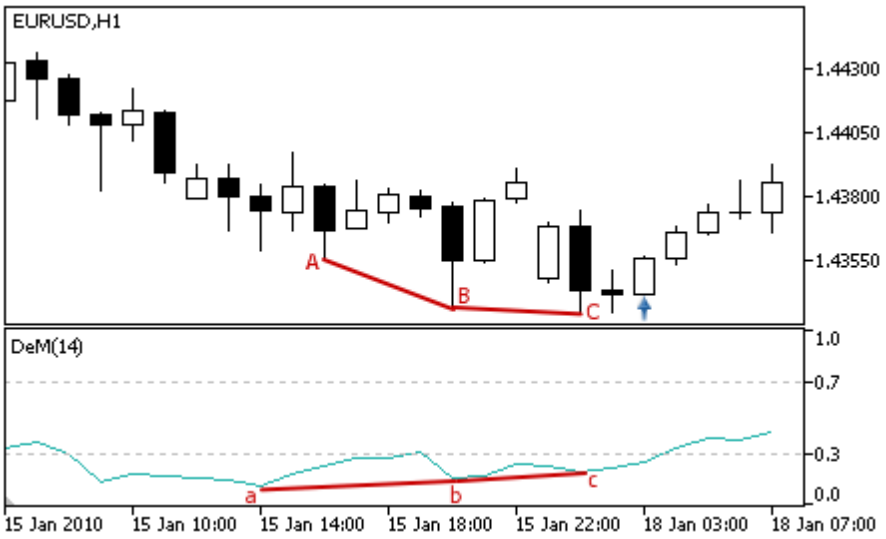
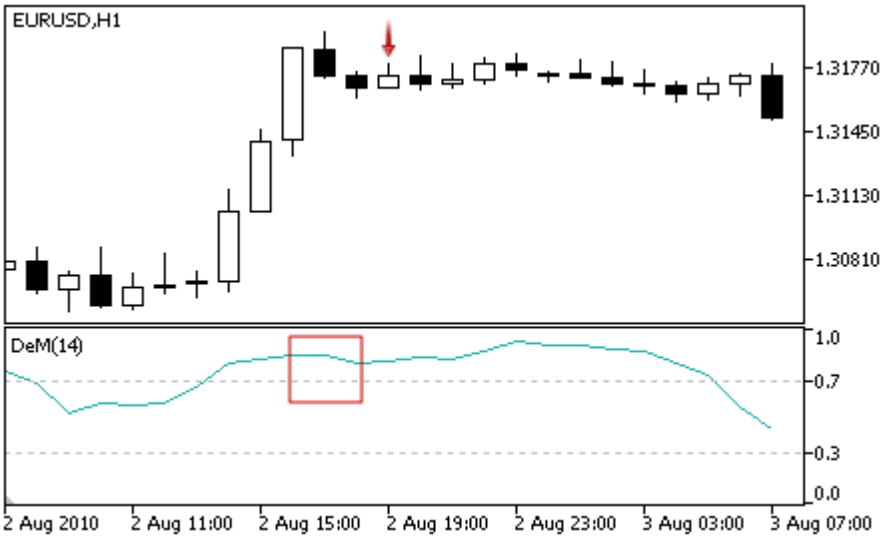
Signals of the Oscillator DeMarker

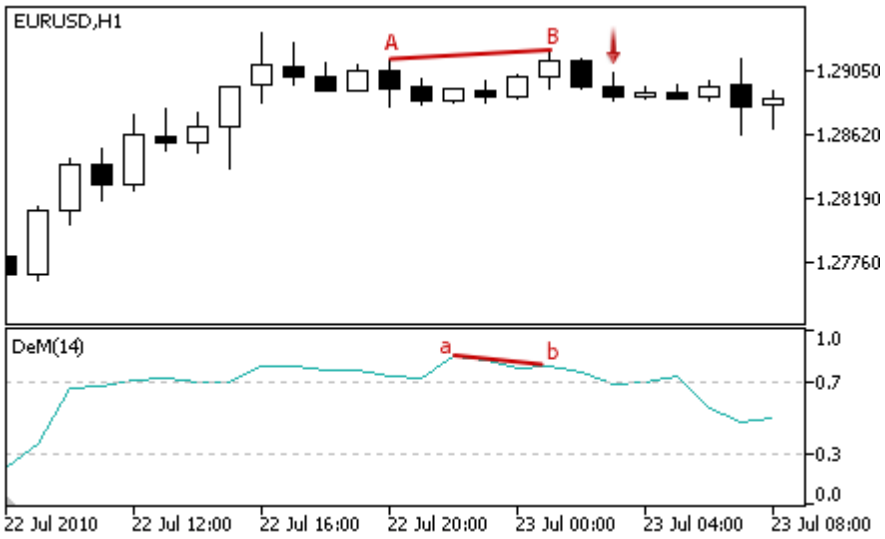
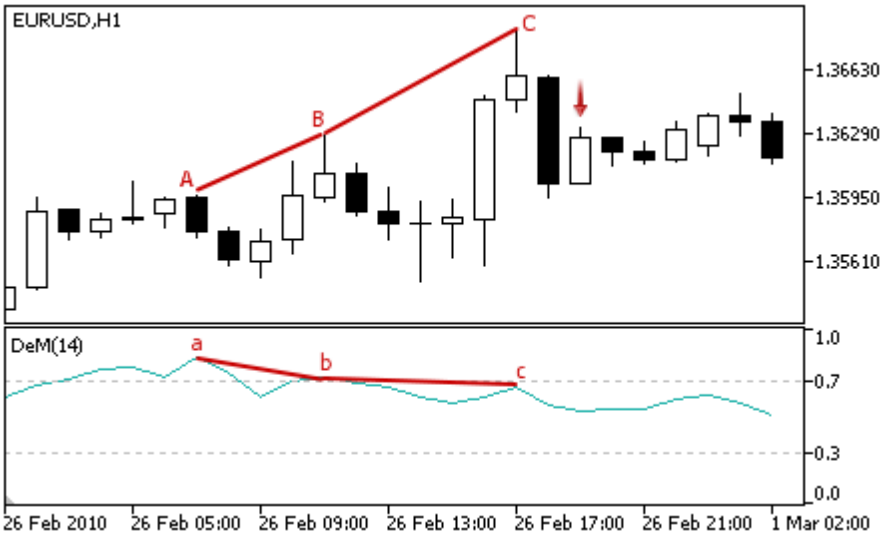
This module of signals is based on the market models of the oscillator DeMarker. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• **Reverse behind the oversold level** — the oscillator turned upwards and its value at the analyzed bar is behind the oversold level (default value is 0.3).  • **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.  • • **Double divergence** — the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding

Signal Type	Description of Conditions
	bottoms, and each of them is lower than the previous one. 
For selling	• **Reverse behind the overbought level** — the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is 0.7).  • **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

Signal Type	Description of Conditions
	 • **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

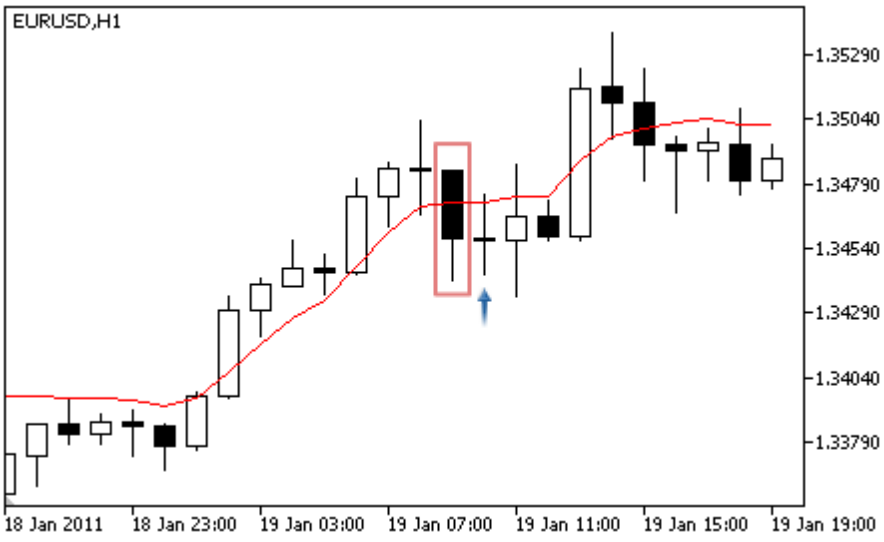
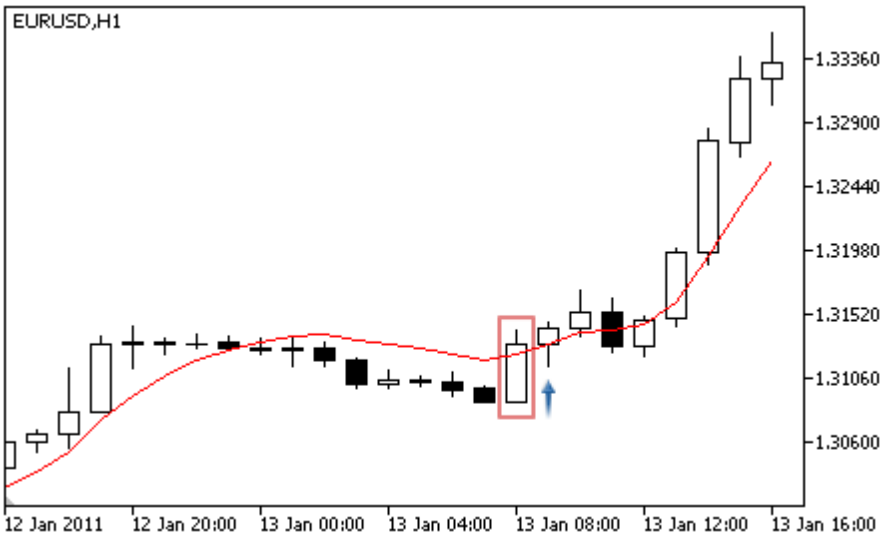
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodDeM	Period of calculation of the oscillator.

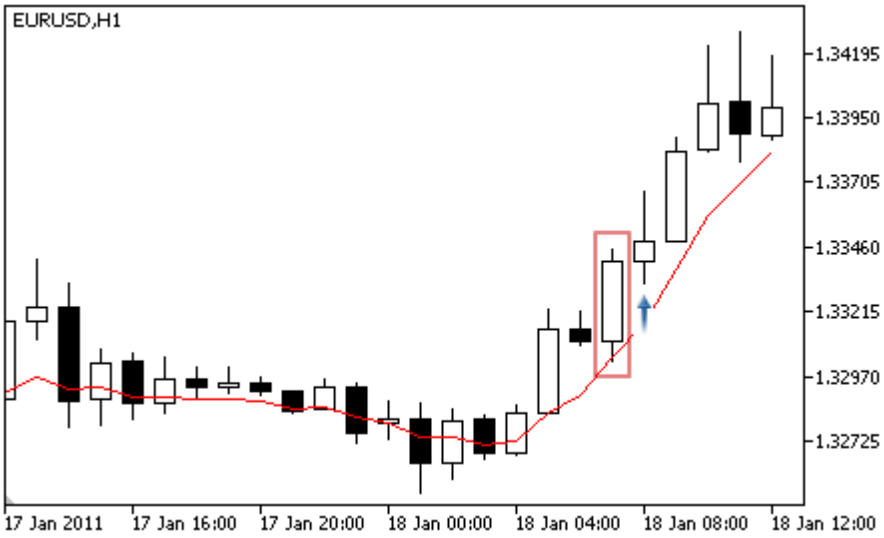
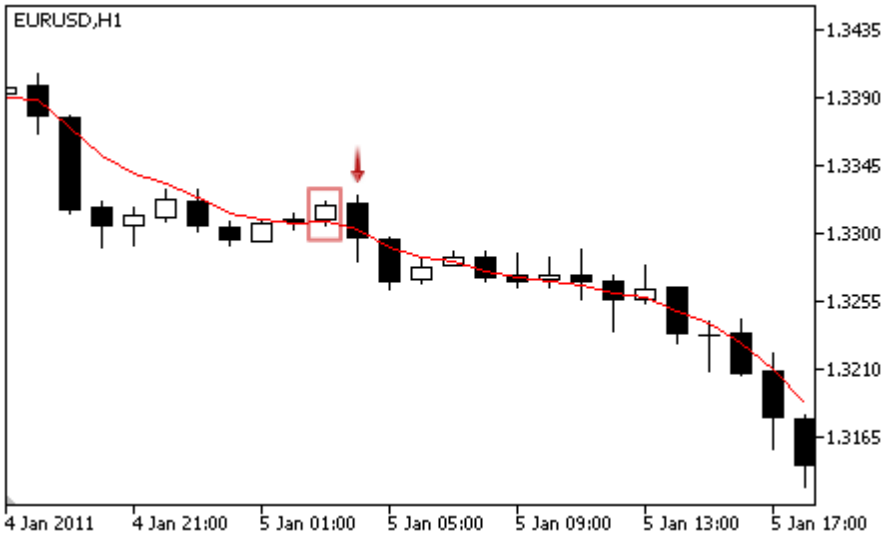
Signals of the Indicator Double Exponential Moving Average

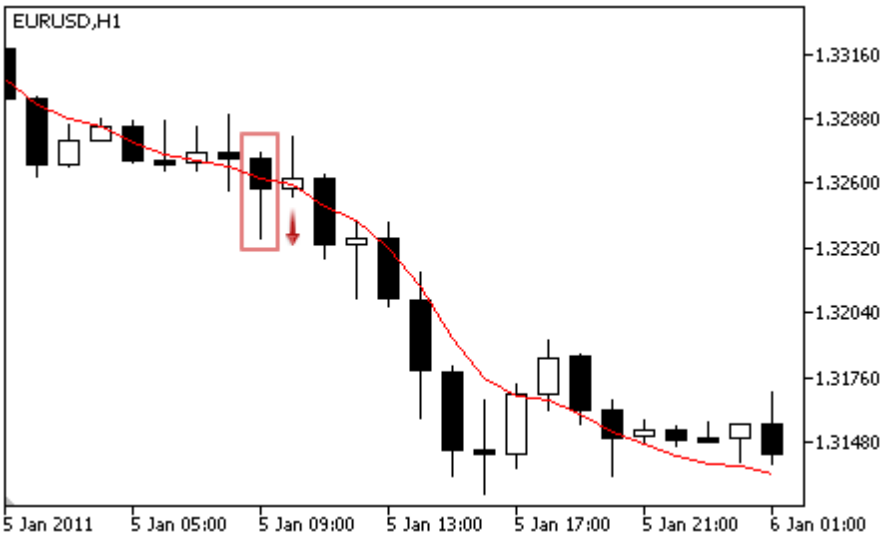
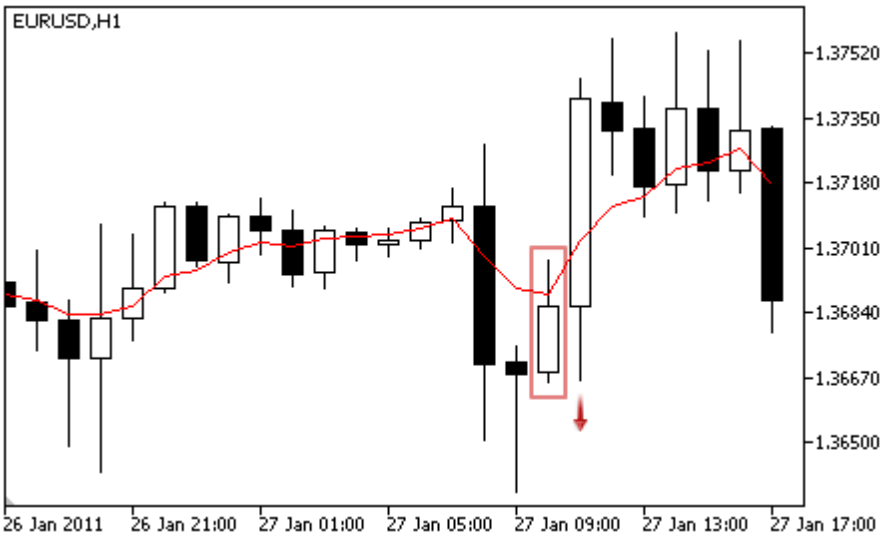
This module is based on the market models of the indicator Double Exponential Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal).  • **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	• The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is above the indicator, and the Low price is below the indicator) and the indicator rises (weak signal). 
For selling	• The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal).  • **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 • The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is below the indicator, and the High price is above the indicator) and the indicator falls (weak signal). 
No objections to buying	The price is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

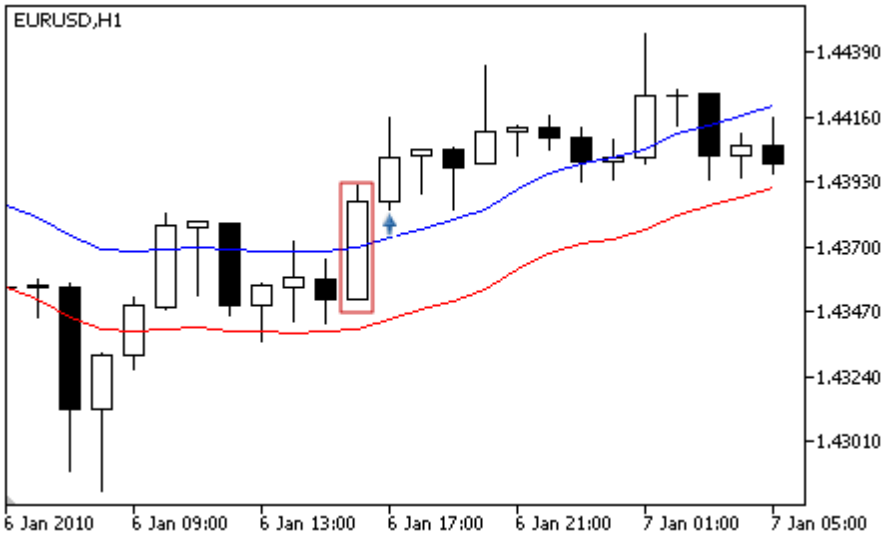
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shit of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

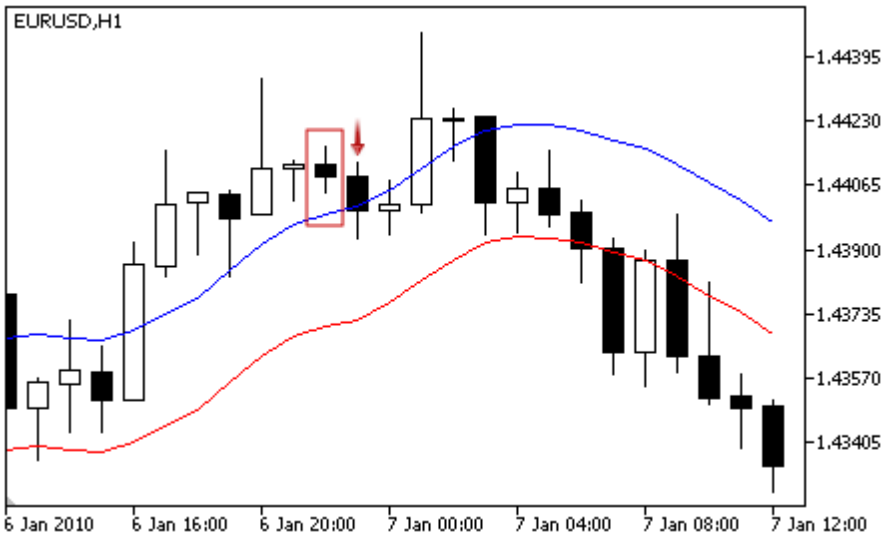
Signals of the Indicator Envelopes

This module of signals is based on the market models of the indicator Envelopes. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• The price is near the lower line of the indicator at the analyzed bar.  • The price crossed the upper line of the indicator at the analyzed bar. 
For selling	• The price is near the upper line of the indicator at the analyzed bar.

Signal Type	Description of Conditions
	 • The price crossed the lower line of the indicator at the analyzed bar. 
No objections to buying	No signals.
No objections to selling	No signals.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

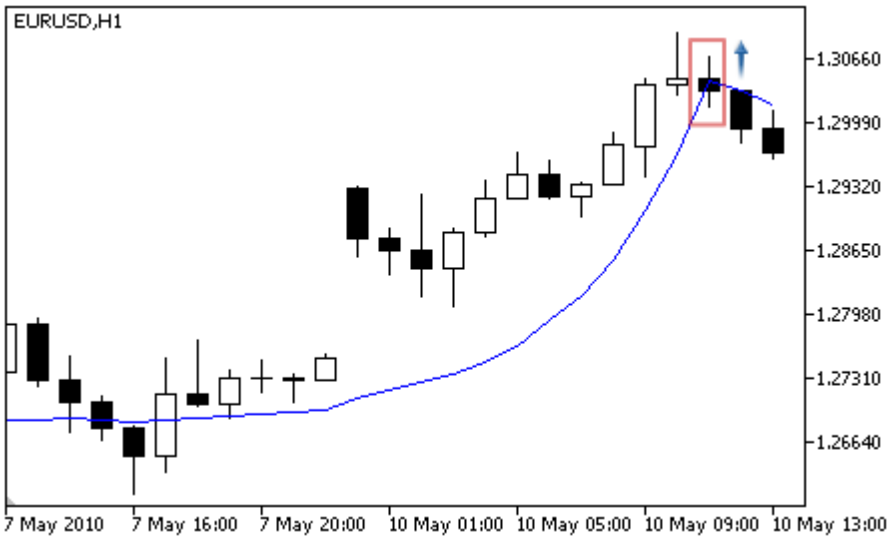
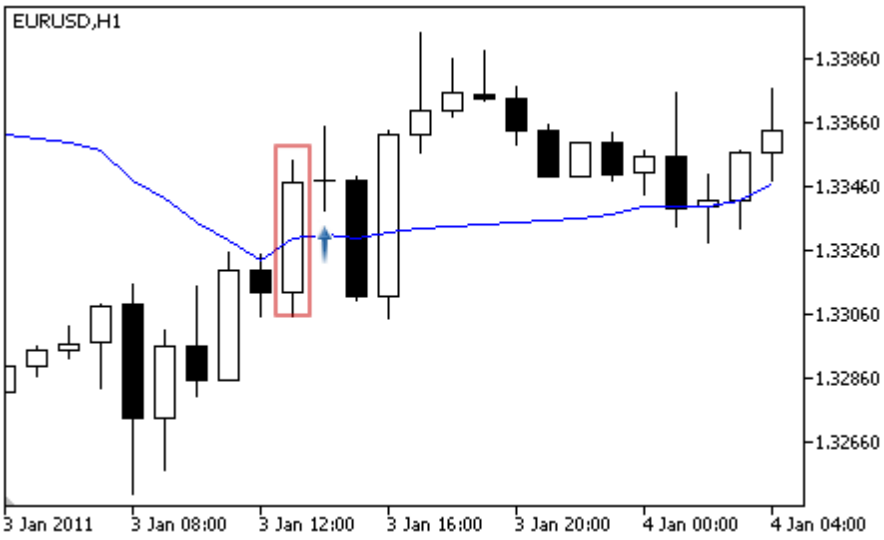
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of calculation of the indicator.
Shift	Shit of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.
Deviation	Deviation of the envelope borders from the center line (MA) in percentage terms.

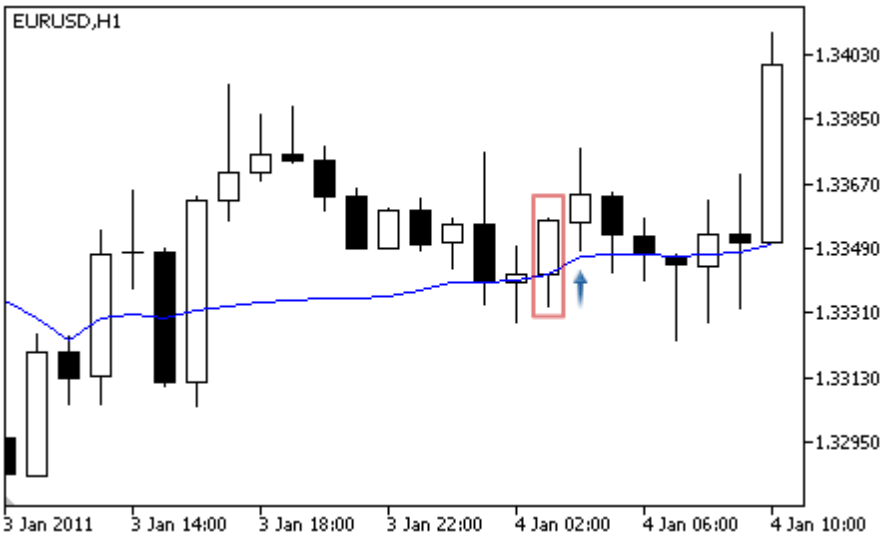
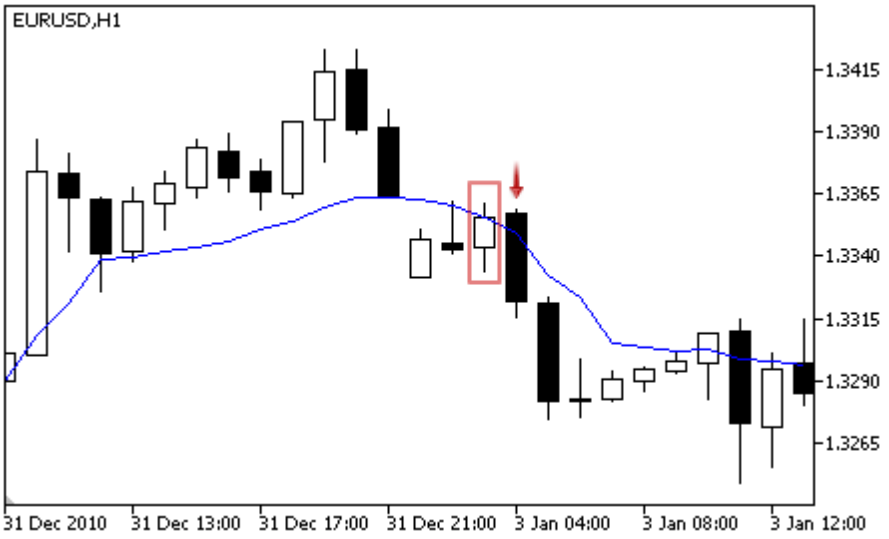
Signals of the Indicator Fractal Adaptive Moving Average

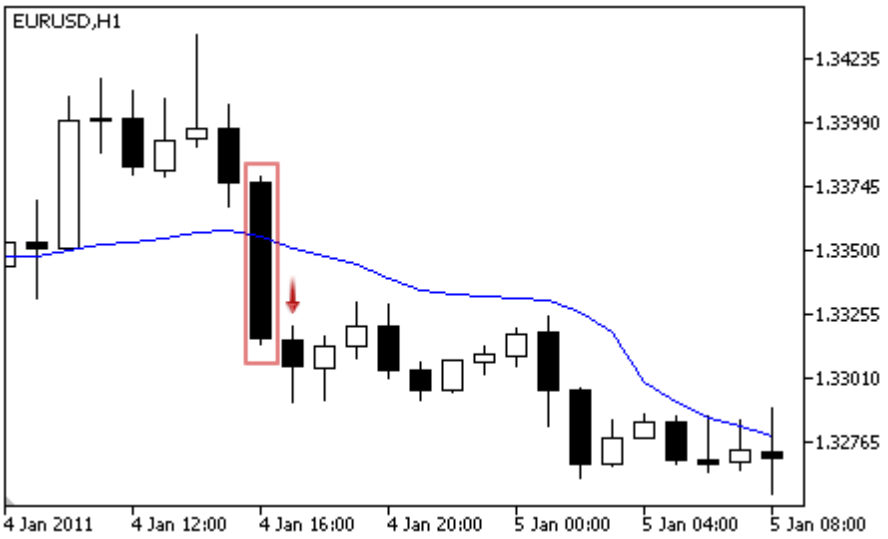
This module of signals is based on the market models of the indicator Fractal Adaptive Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

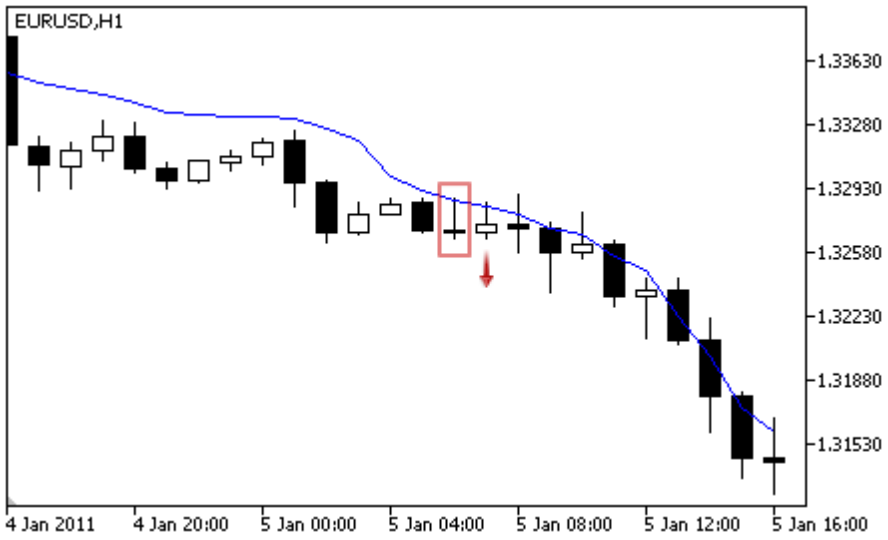
Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal).  • **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	• The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is above the indicator, and the Low price is below the indicator) and the indicator rises (weak signal). 
For selling	• The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal).  • **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

<table>
<tr><th>Signal Type</th><th>Description of Conditions</th></tr>
<tr><td></td><td> 

The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is below the indicator, and the High price is above the indicator) and the indicator falls (weak signal).

</td></tr>
<tr><td>No objections to buying</td><td>The prices is above the indicator.</td></tr>
<tr><td>No objections to selling</td><td>The price is below the indicator.</td></tr>
</table>

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shit of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

Signals of the Intraday Time Filter

This module is based on the assumption that the efficiency of market models changes in time. Using this module, you can filter signals received from the other modules by hour and days of week. It allows increasing the quality of generated signals due to cutting off the unfavorable time periods. The mechanism of making trade decisions on the basis of signals of the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	No signals.
For selling	No signals.
No objections to buying	The current date and time meet the specified parameters.
No objections to selling	The current date and time meet the specified parameters.

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
GoodHourOfDay	Number of the only hour of day (from 0 to 23) when trade signals will be enabled. If the value is -1, the signals will be enabled through the whole day.
BadHoursOfDay	The bit field. Each bit of this field corresponds to an hour of day (0 bit - 0 hour, ..., 23 bit - 23-rd hour). If the value of a bit is equal to 0, trade signals will be enabled during the corresponding hour. If the value of a bit is equal to 1, trade signals will be disabled during the corresponding hour. A specified number is represented as a binary number and is used as bit mask. Disabled hours have higher priority than the enabled ones.
GoodDayOfWeek	Number of the only day of week (from 0 to 6, where 0 is Sunday), when trade signals will be

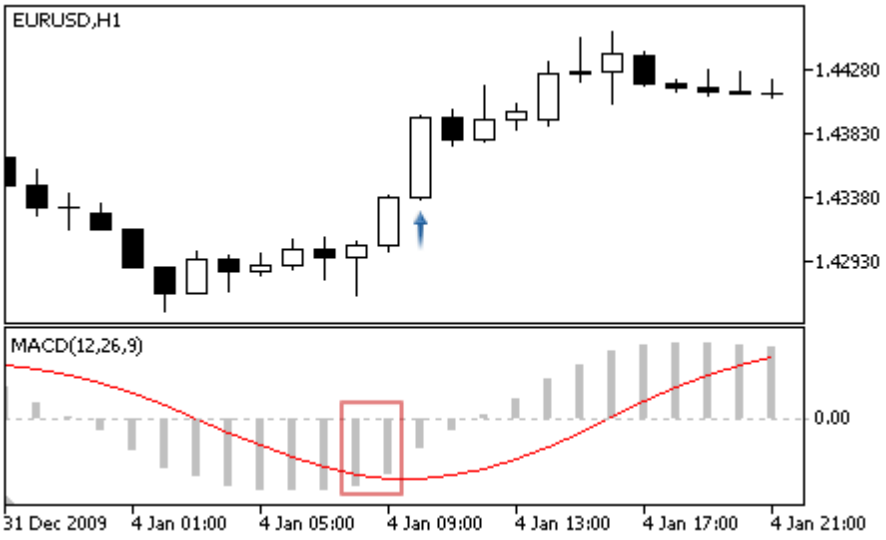
Parameter	Description
	enabled. If the value is -1, the signals will be enabled on any day.
BadDaysOfWeek	The bit field. Each bit of this field corresponds to a day of week (0 bit - Sunday, ..., 6 bit - Saturday). If the value of a bit is equal to 0, trade signals will be enabled during the corresponding day. If the value of a bit is equal to 1, trade signals will be disabled during the corresponding day. A specified number is represented as a binary number and is used as bit mask. Disabled days have higher priority than the enabled ones.

Signals of the Oscillator MACD

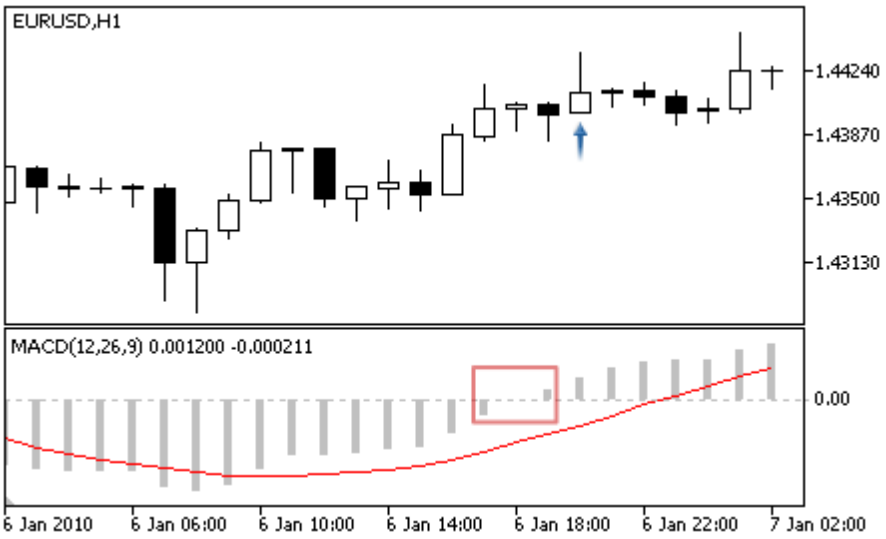
This module of signals is based on the market models of the oscillator MACD. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

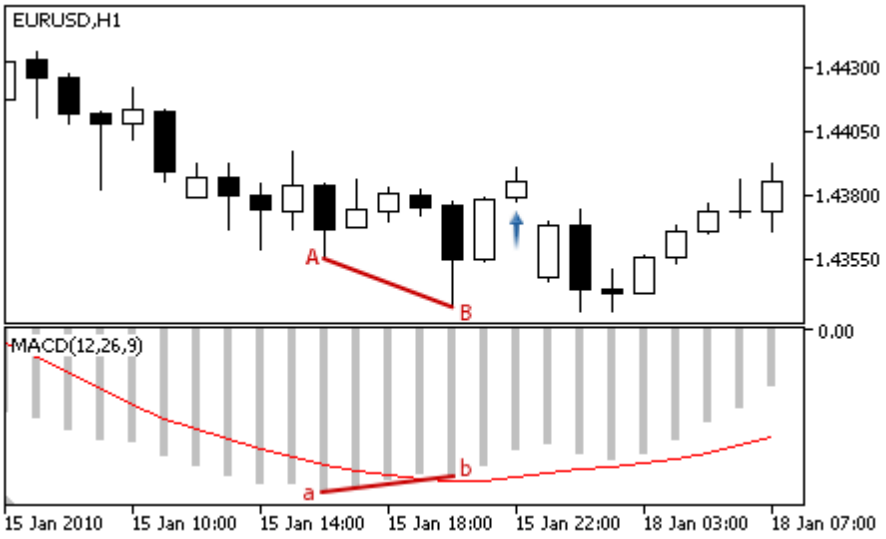
Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• **Reverse** — the oscillator turned upwards (the oscillator rises at the analyzed bar and falls at the previous one).  • **Crossover of the main and signal line** — the main line is above the signal line at the analyzed bar and below the signal line at the previous one.  • **Crossing the zero level** — the main line is above the zero level at the analyzed bar and below the zero level at the previous one.

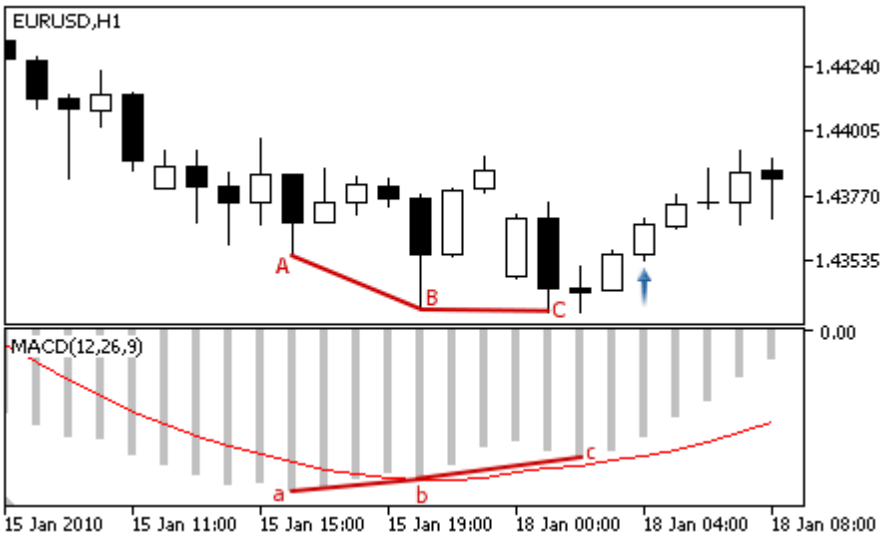
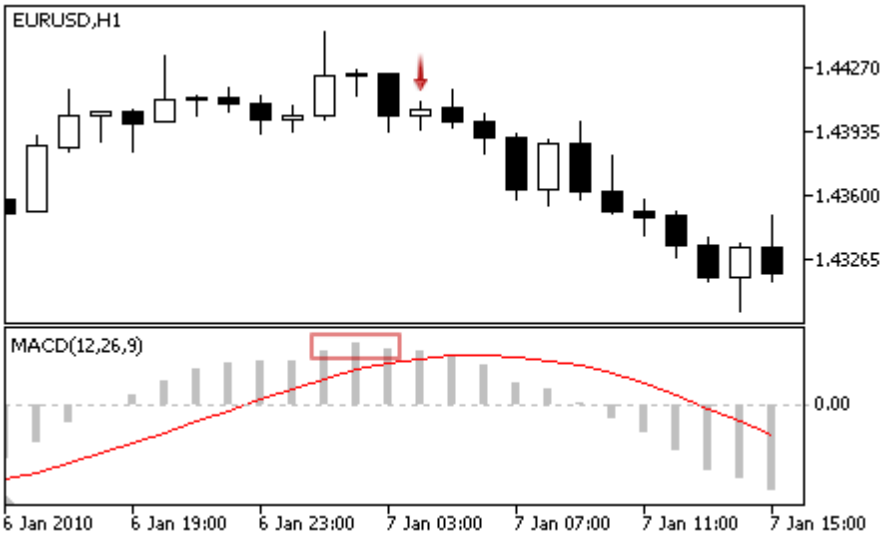
Signal Type	Description of Conditions



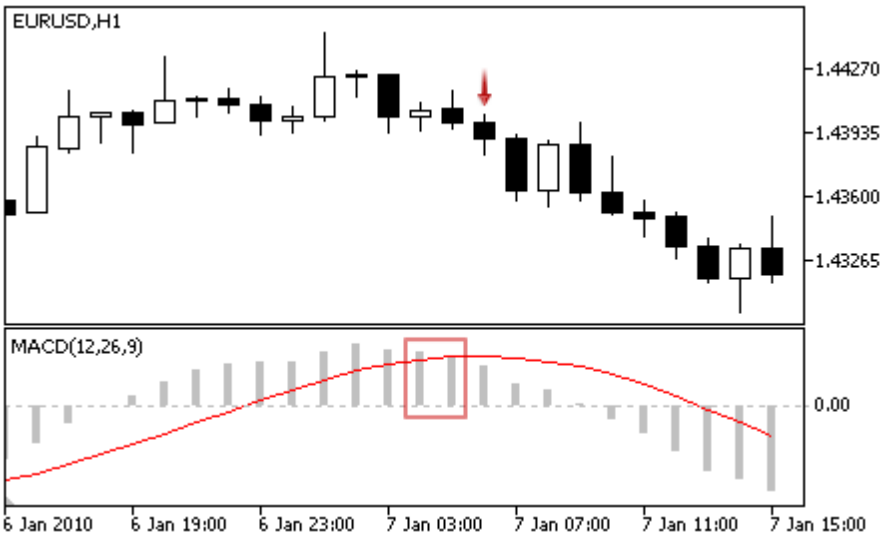
- **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.



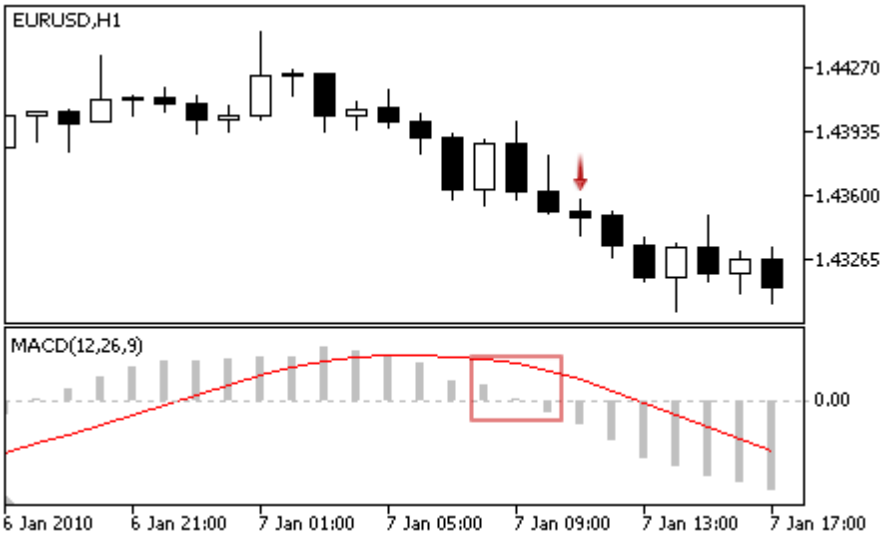
- **Double divergence** — the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one.

Signal Type	Description of Conditions
	
For selling	• **Reverse** — the oscillator turned downwards (the oscillator falls at the analyzed bar and rises at the previous one).  • **Crossover of the main and signal line** — the main line is below the signal line at the analyzed bar and above the signal line at the previous one.

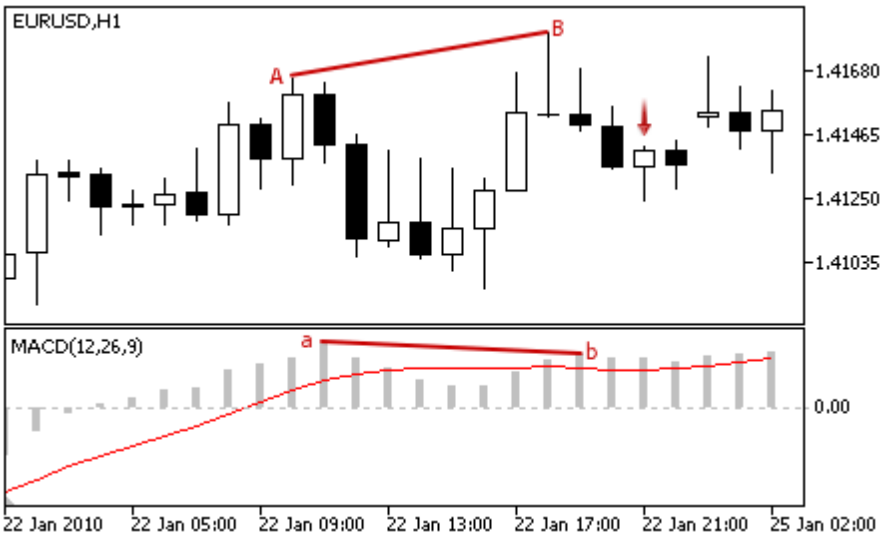
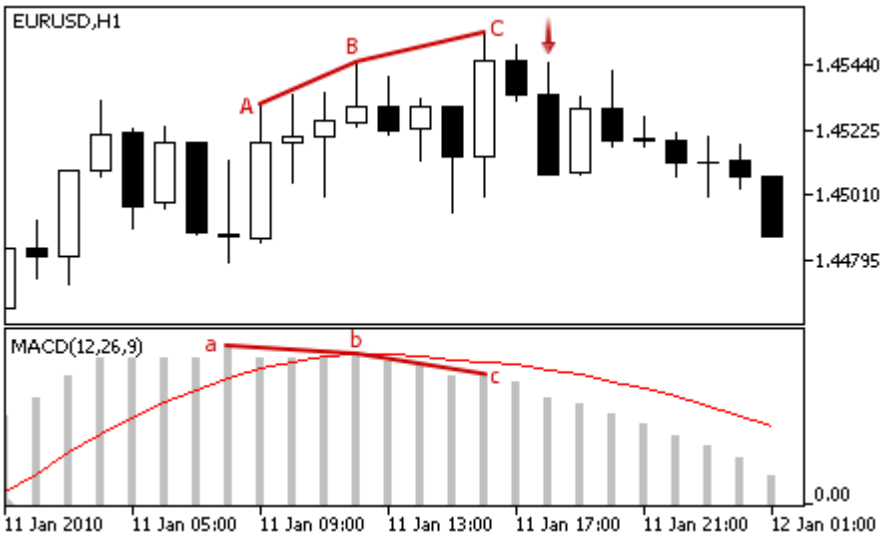
Signal Type	Description of Conditions



- **Crossing the zero level** — the main line is below the zero level at the analyzed bar and above the zero level at the previous one.



- **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

Signal Type	Description of Conditions
	 • **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

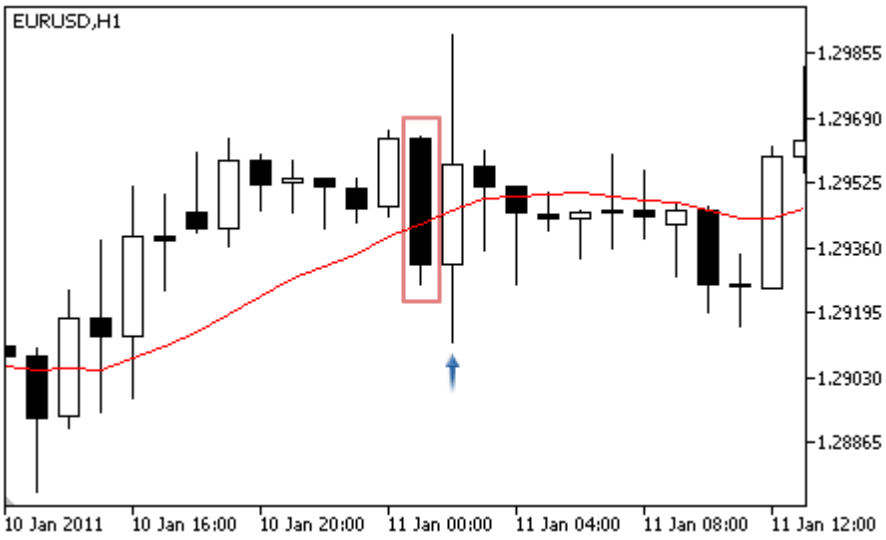
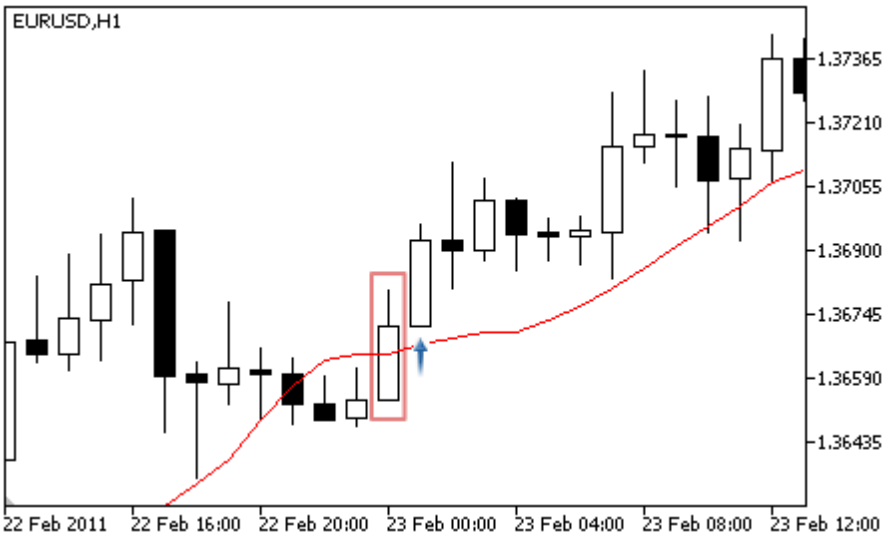
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodFast	Period of calculation of the fast EMA.
PeriodSlow	Period of calculation of the slow EMA.
PeriodSignal	Period of smoothing.
Applied	A price series used for calculation of the oscillator.

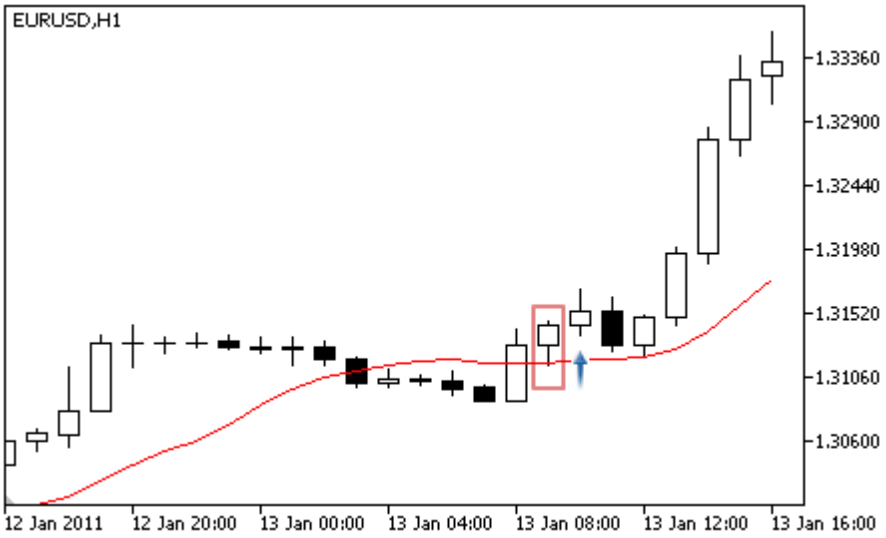
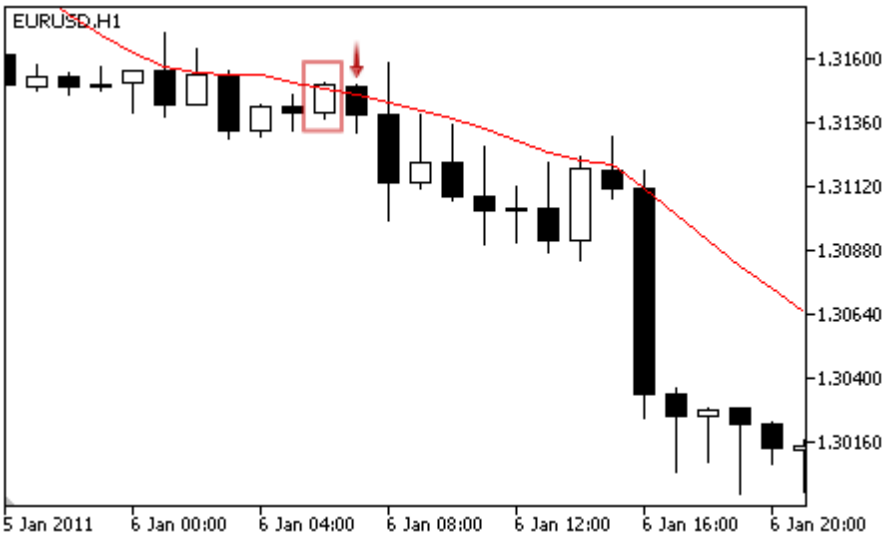
Signals of the Indicator Moving Average

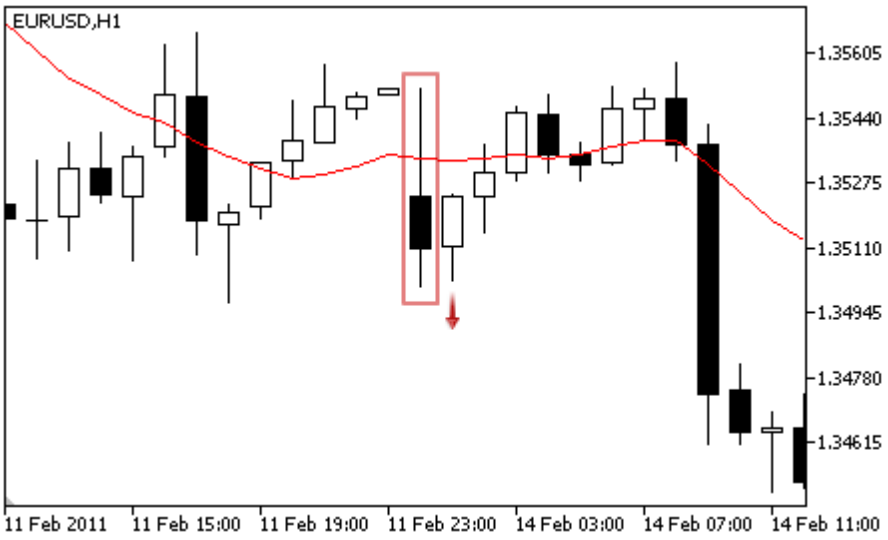
This module of signals is based on the market models of the indicator Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal).  • **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	• The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is above the indicator, and the Low price is below the indicator) and the indicator rises (weak signal). 
For selling	• The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal).  • **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 • The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is below the indicator, and the High price is above the indicator) and the indicator falls (weak signal). 
No objections to buying	The prices is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

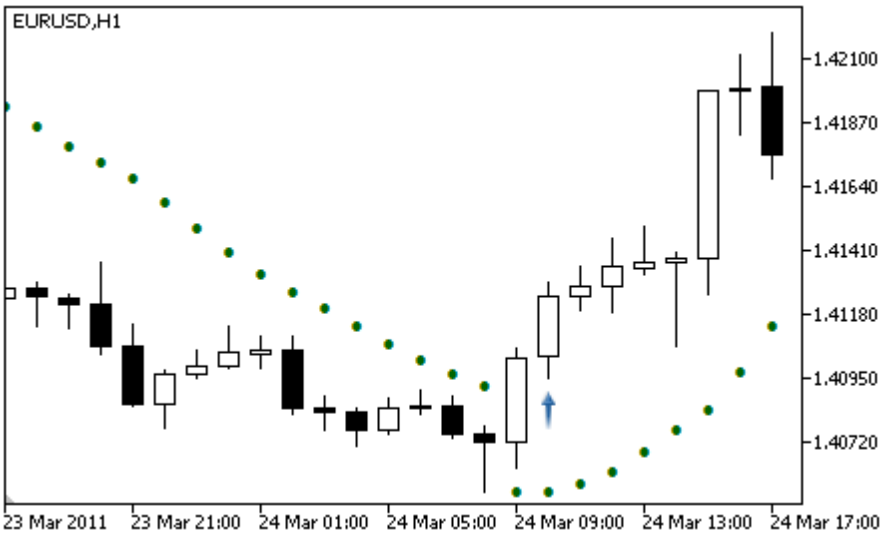
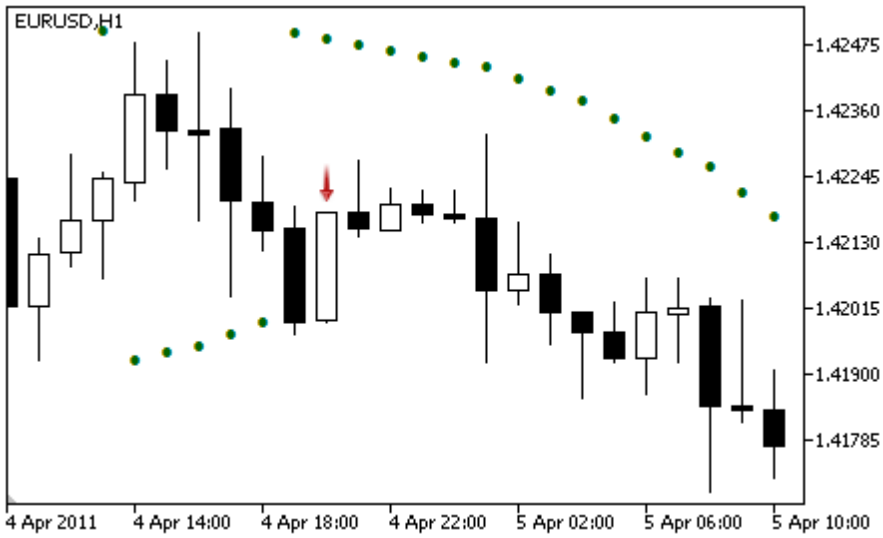
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shit of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

Signals of the indicator Parabolic SAR

This module of signals is based on the market models of the indicator Parabolic SAR. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	**Reverse** — the indicator is below the price at the analyzed bar and above the price at the previous one. 
For selling	**Reverse** — the indicator is above the price at the analyzed bar and below the price at the previous one. 
No objections	The prices is above the indicator.

Signal Type	Description of Conditions
to buying	
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

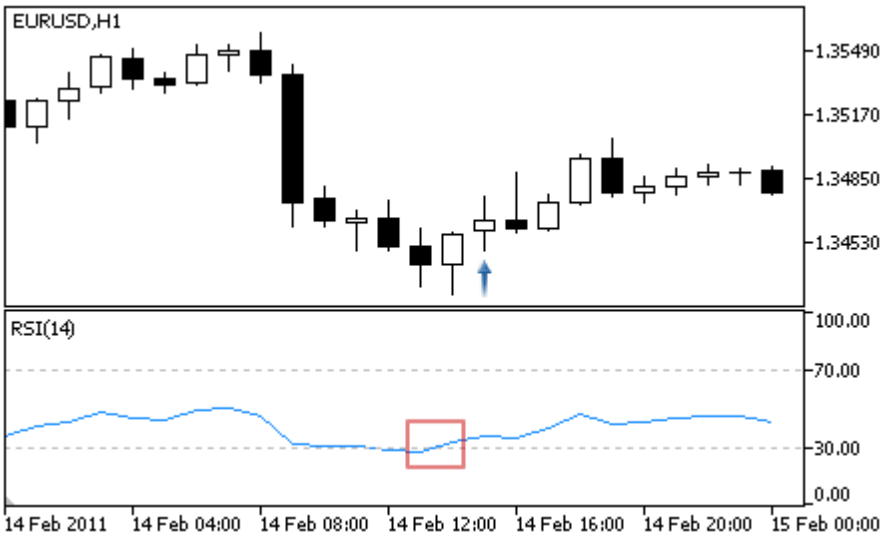
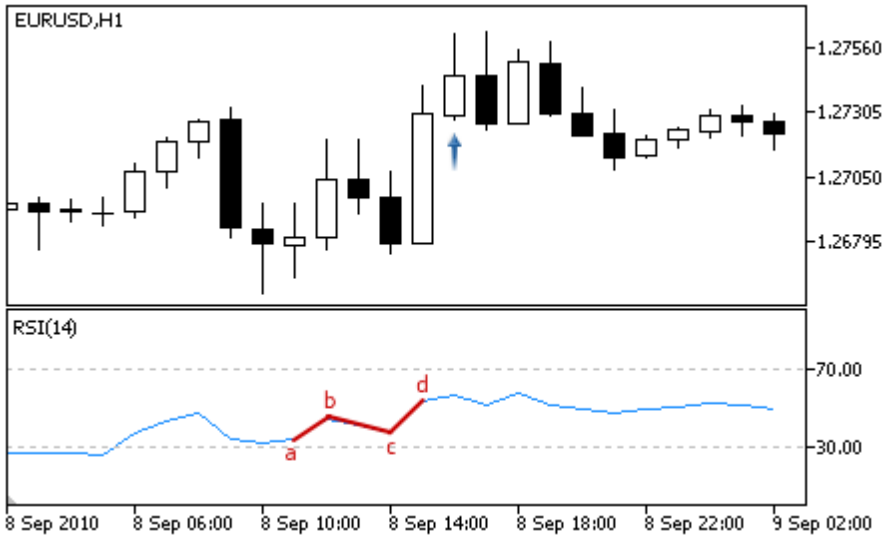
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
Step	The increment of speed of the indicator.
Maximum	Maximum rate of the speed of convergence of the indicator with the price.

Signals of the Oscillator Relative Strength Index

This module of signals is based on the market models of the oscillator Relative Strength Index. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

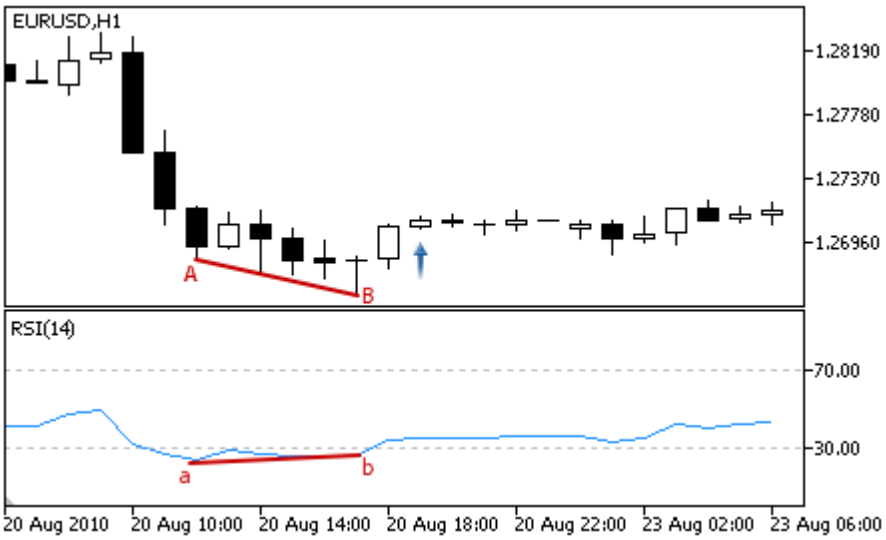
Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

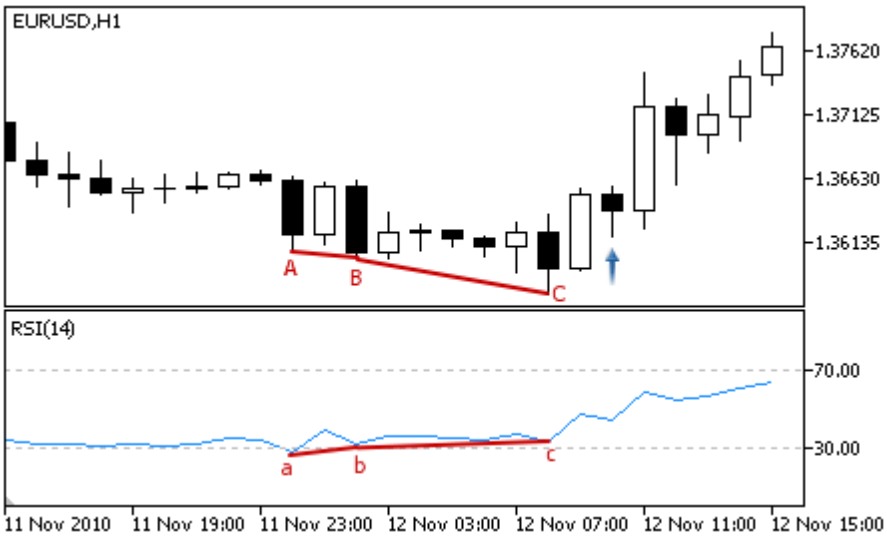
Signal Type	Description of Conditions
For buying	• **Reverse behind the oversold level** — the oscillator turned upwards and its value at the analyzed bar is behind theo oversold level (default value is 30).  • **Failed swing** — the oscillator rises higher than the previous peak at the analyzed bar.  • **Divergence** — the first analyzed bottom of the oscillator is lower than the previous

Signal Type	Description of Conditions

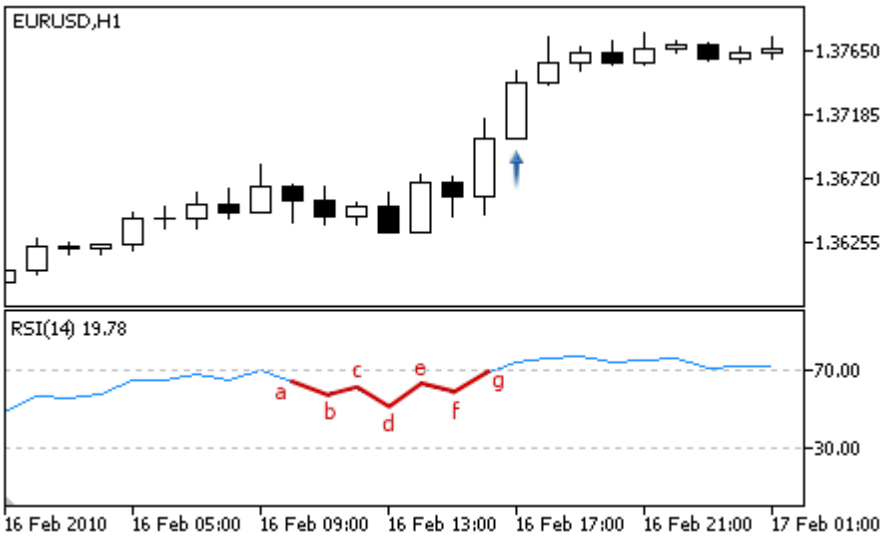
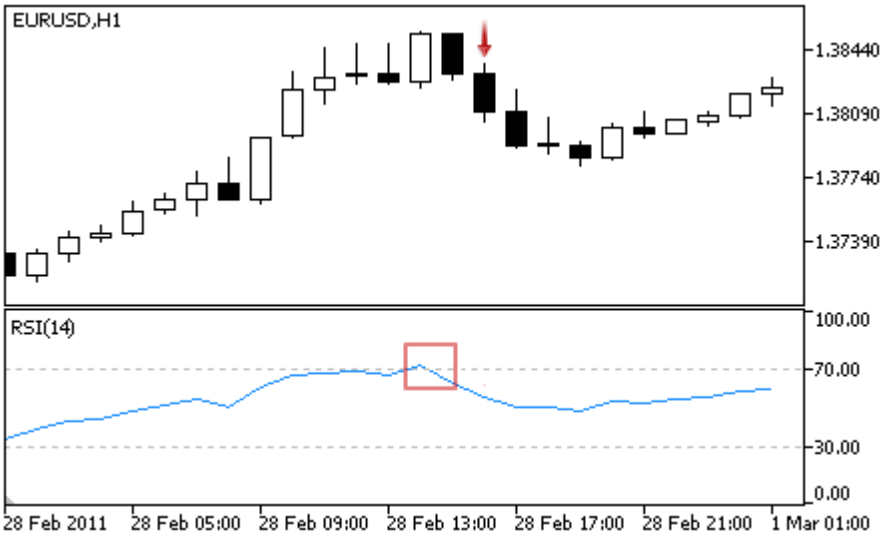
one, and the corresponding price bottom is lower than the previous one.



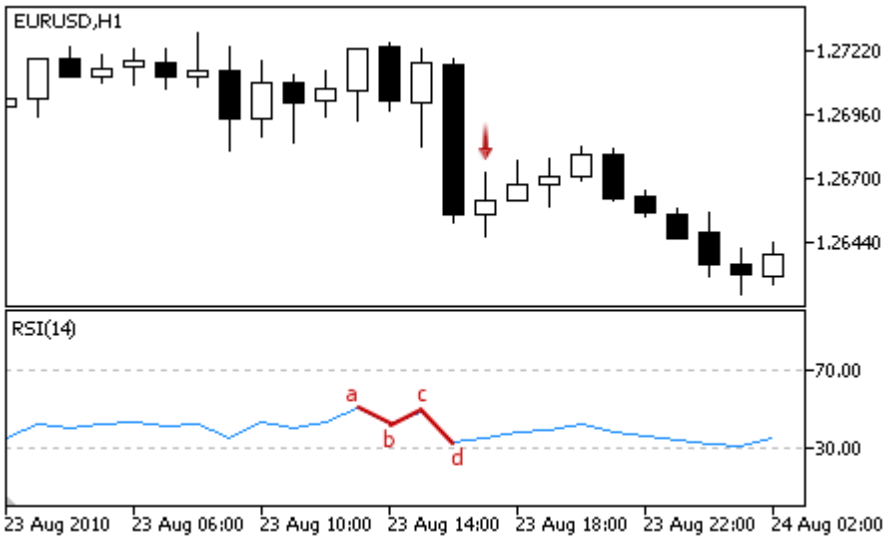
- **Double divergence** — the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one.



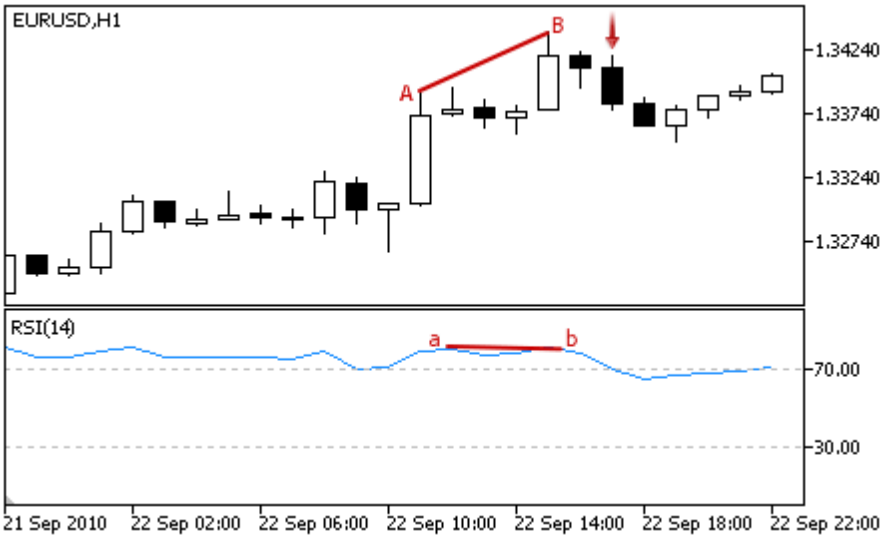
- **Head/Shoulders** — the oscillator formed three consequent bottoms, and the mid one is lower than the others.

Signal Type	Description of Conditions
	
For selling	• **Reverse behind the overbought level** — the oscillator turned downwards and its value at the analyzed bar is behind the overbought level (default value is 70).  • **Failed swing** — the oscillator falls lower than the previous bottom at the analyzed bar.

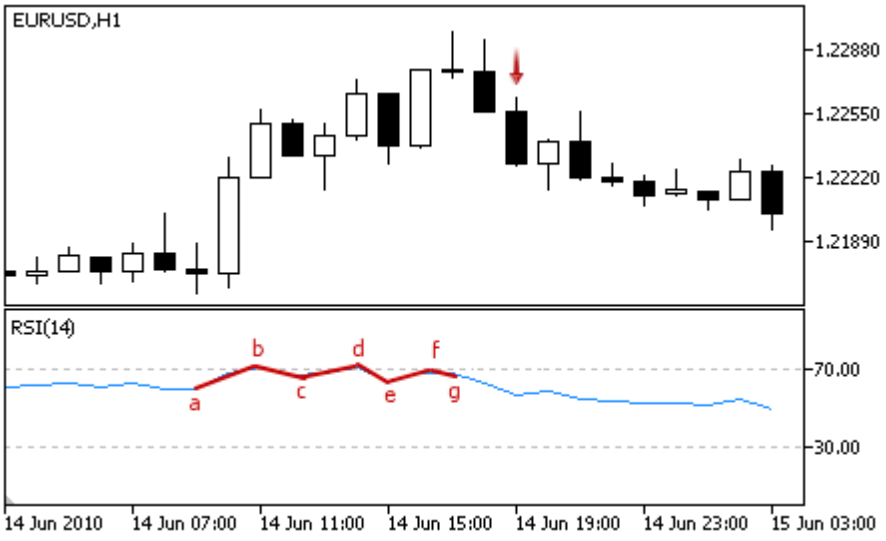
Signal Type	Description of Conditions



- **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.



- **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one.

Signal Type	Description of Conditions
	 • **Head/Shoulders** — the oscillator formed three consequent peaks, and the mid one is higher than the others. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

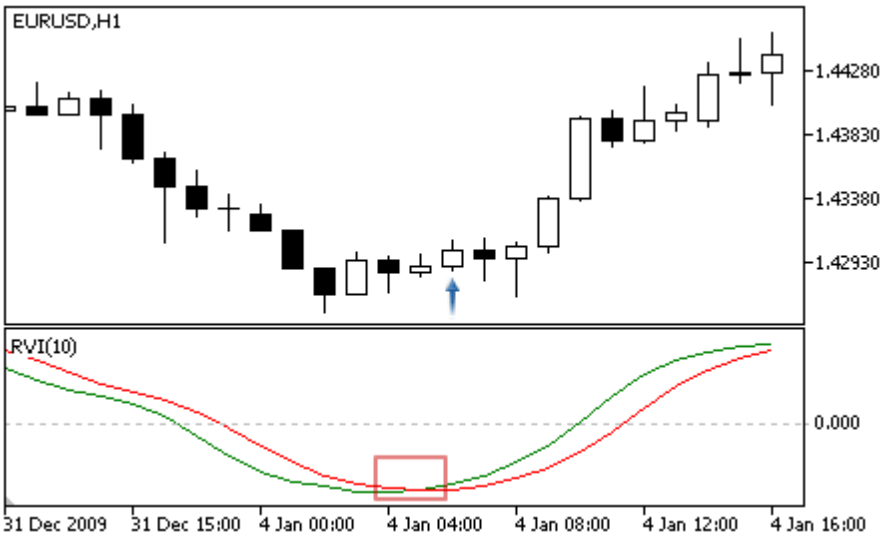
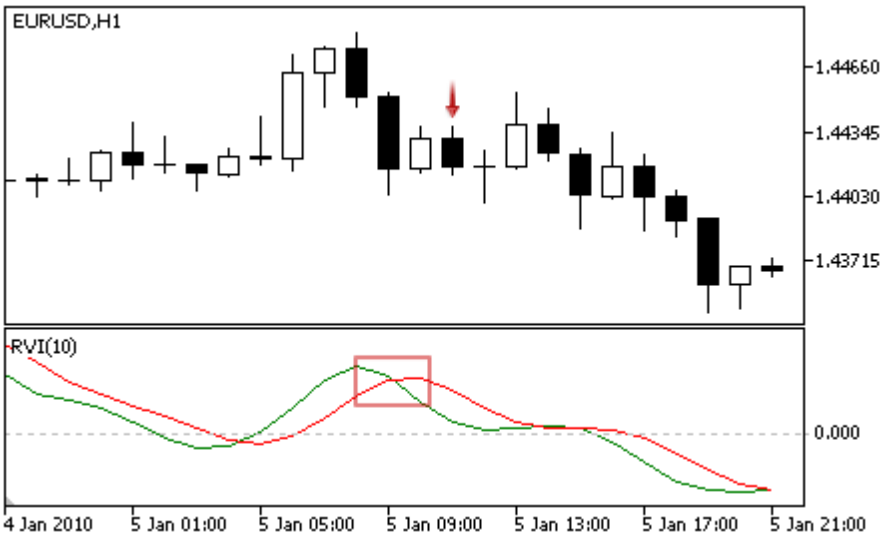
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodRSI	Period of calculation of the oscillator.
Applied	A price series used for calculation of the oscillator.

Signals of the Oscillator Relative Vigor Index

This module of signals is based on the market models of the oscillator Relative Vigor Index. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	**Crossing of the main and signal line** — the main line is above the signal line at the analyzed bar and below the signal line at the previous one. 
For selling	**Crossing of the main and signal line** — the main line is below the signal line at the analyzed bar and above the signal line at the previous one. 
No objections	Value of the oscillator grows at the analyzed bar.

Signal Type	Description of Conditions
to buying	
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

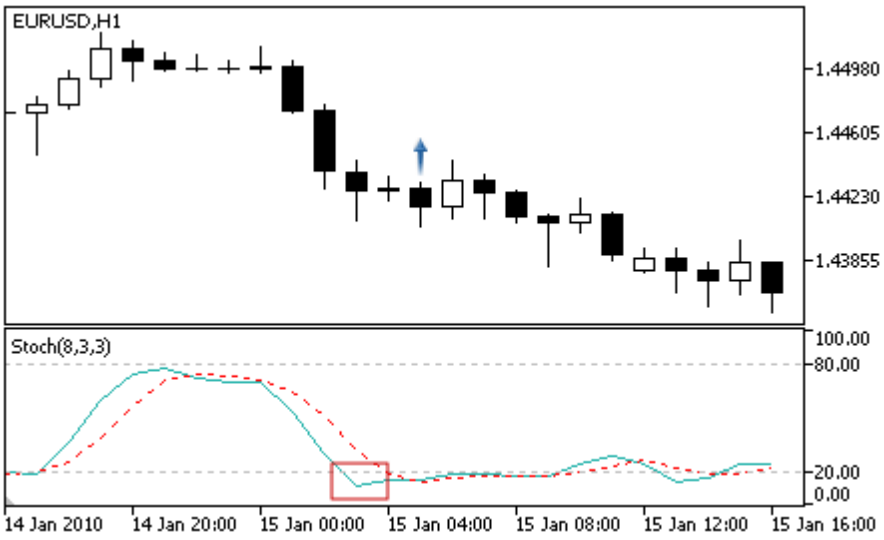
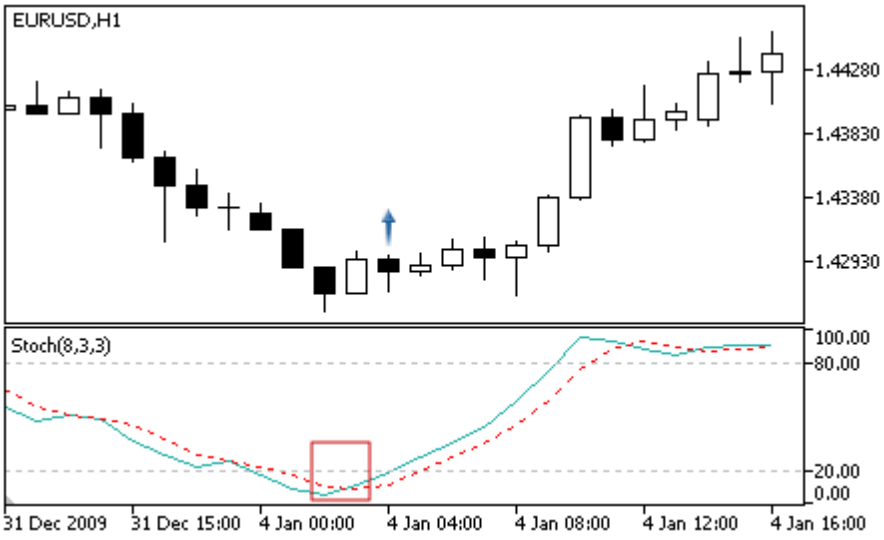
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodRVI	Period of calculation of the oscillator.

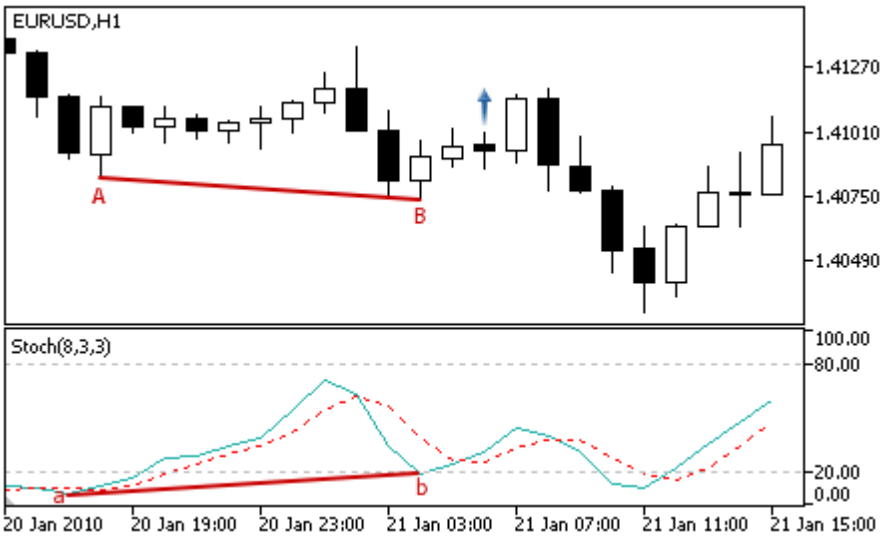
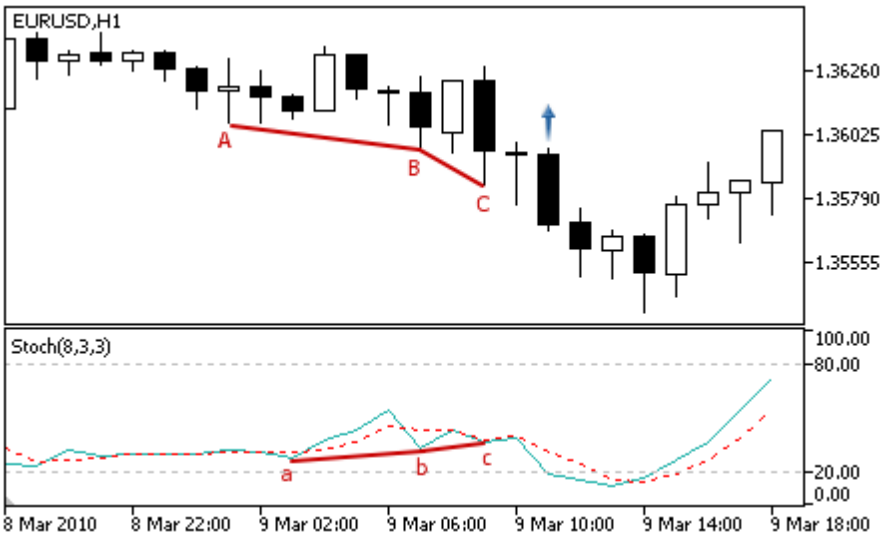
Signals of the Oscillator Stochastic

This module of signals based on the market models of the oscillator Stochastic. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

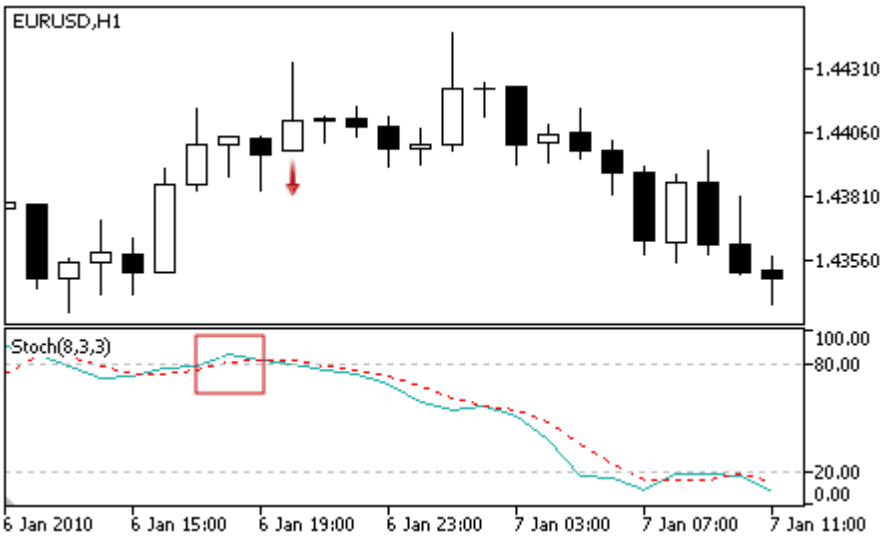
Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

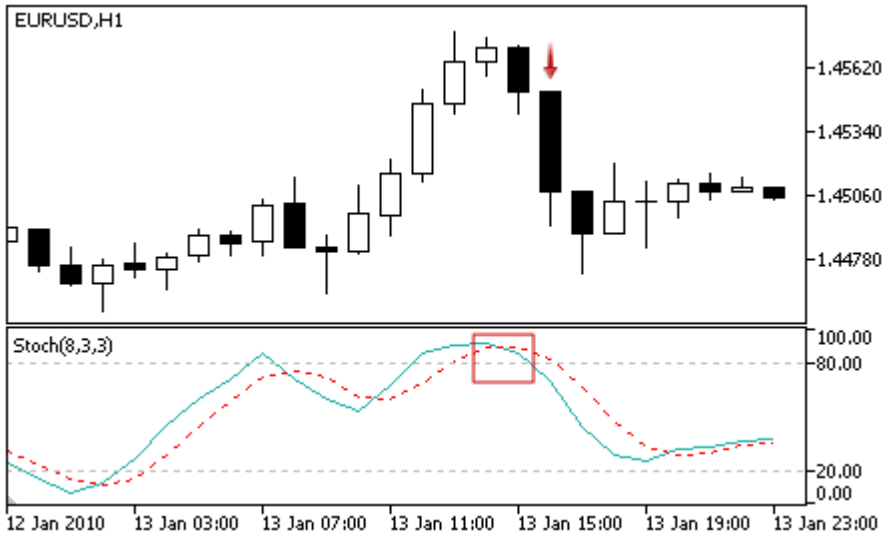
Signal Type	Description of Conditions
For buying	• **Reverse** — the oscillator turned upwards (the oscillator rises at the analyzed bar and falls at the previous one).  • **Crossing of the main and signal line** — the main line is above the signal line at the analyzed bar and below the signal line at the previous one.  • **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one.

Signal Type	Description of Conditions
	 • **Double divergence** — the oscillator form three consequent bottoms, each of them is higher than the previous one; and the price formed three corresponding bottoms, and each of them is lower than the previous one. 
For selling	• **Reverse** — the oscillator turned downwards (the oscillator falls at the analyzed bar and rises at the previous one).

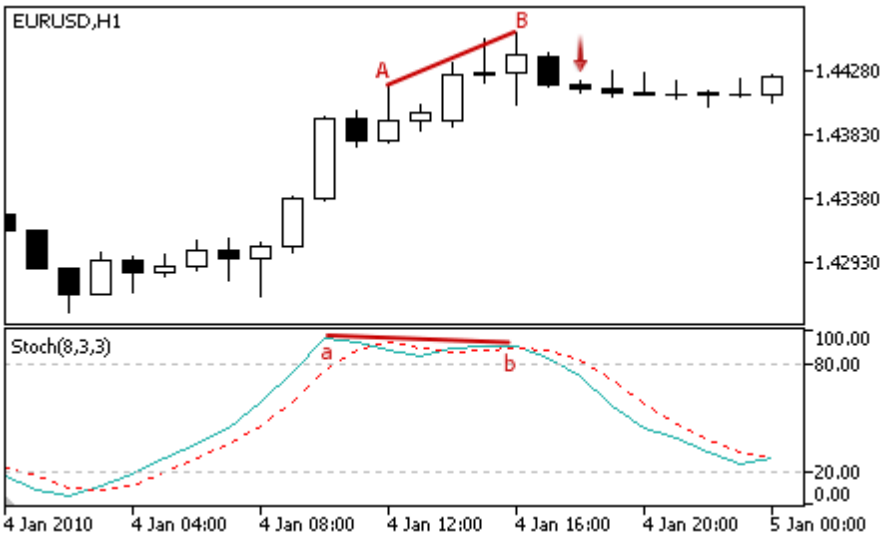
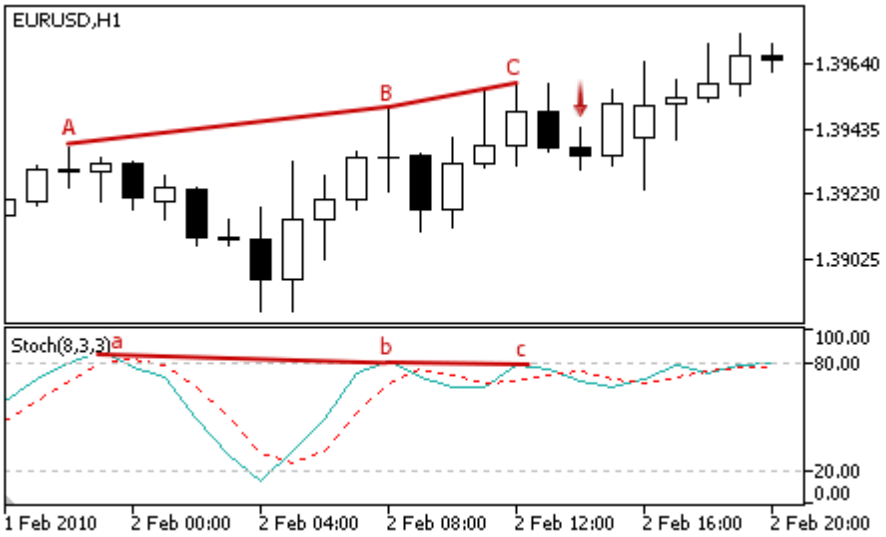
Signal Type	Description of Conditions



- **Crossing of the main and signal line** — the main line is below the signal line at the analyzed bar and above the signal line at the previous one.



- **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak.

Signal Type	Description of Conditions
	 • **Double divergence** — the oscillator formed three consequent peaks, each of them is lower than the previous one; and the price formed three corresponding peaks, each of them is higher than the previous one. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

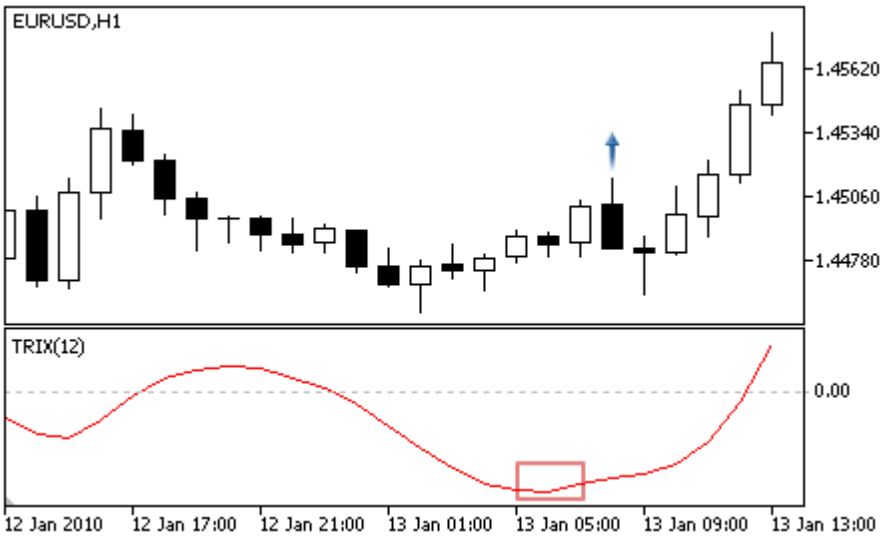
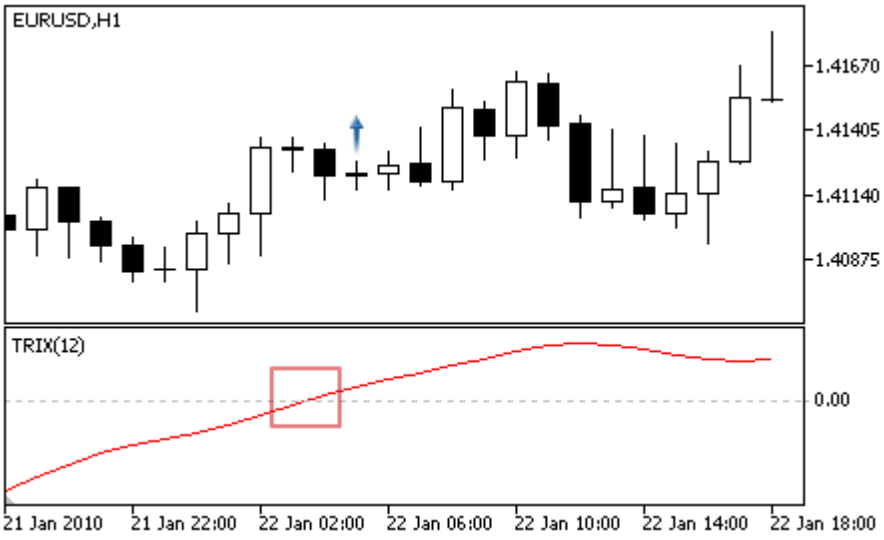
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodK	Period of calculation of the main line of the oscillator.
PeriodD	Period of calculation of the main line of the oscillator.
PeriodSlow	Period of slowing.
Applied	A price series used for calculation of the oscillator.

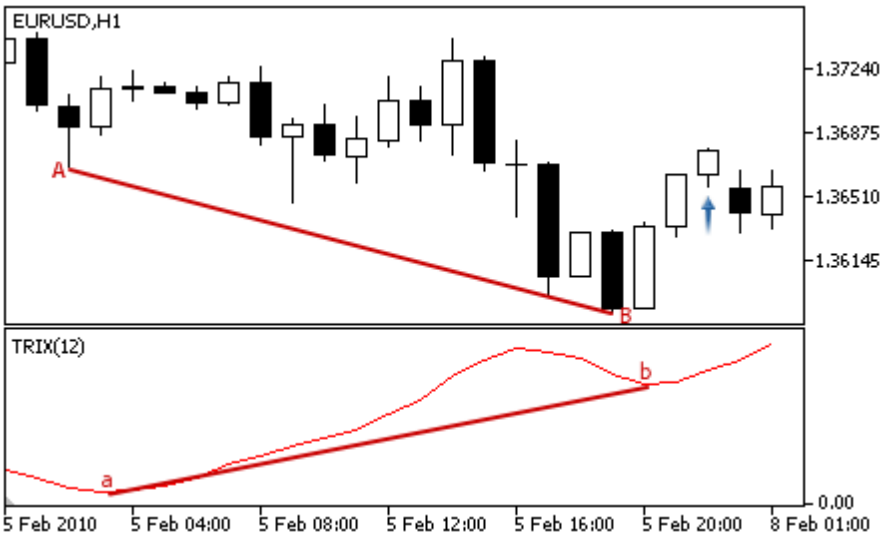
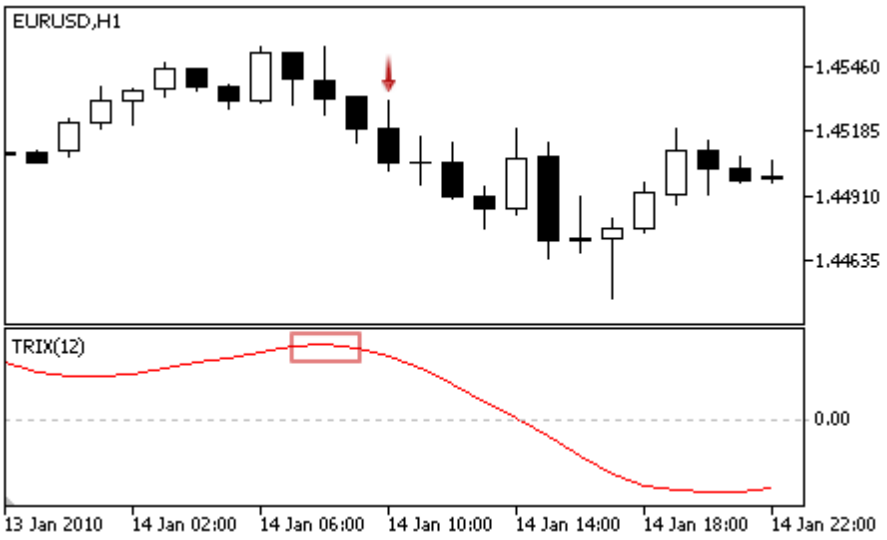
Signals of the Oscillator Triple Exponential Average

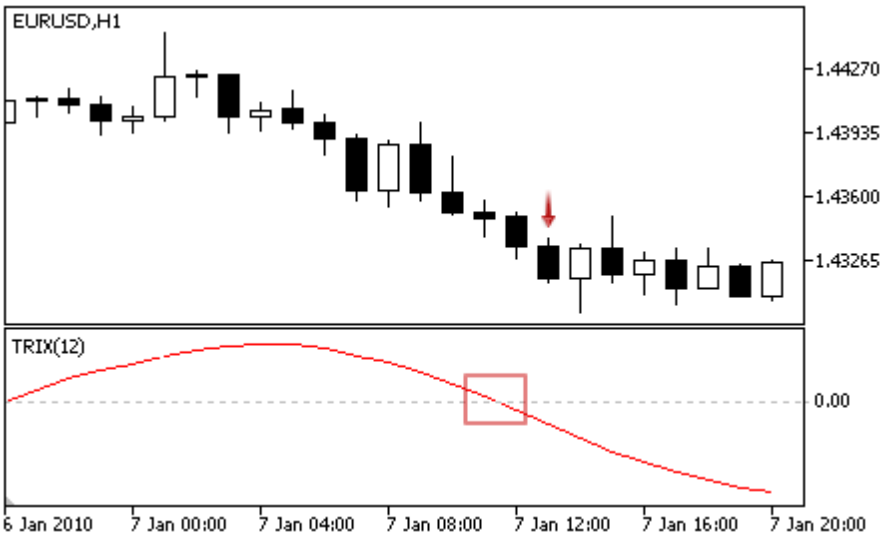
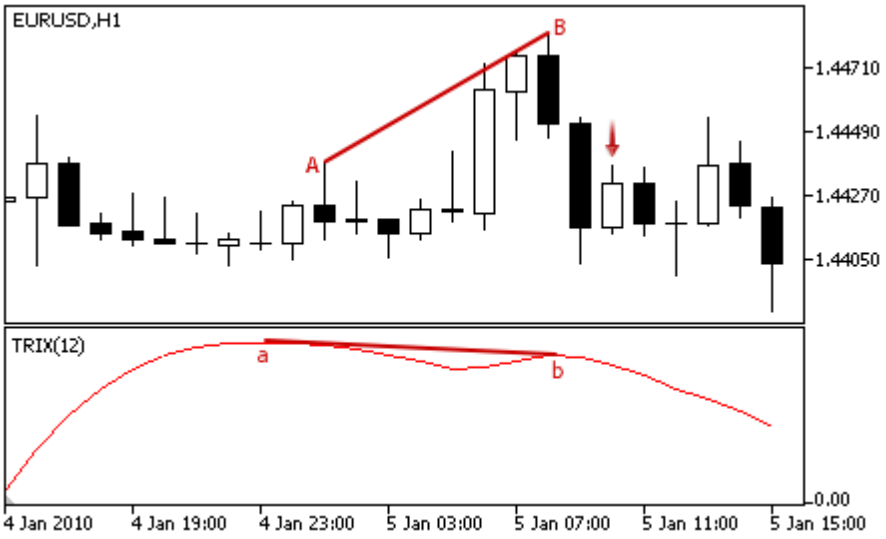
This module of signals is based on the market models of the oscillator Triple Exponential Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• **Reverse** — the oscillator turned upwards (the oscillator rises at the analyzed bar and falls at the previous one).  • **Crossing the zero level** — the main line is above the zero level at the analyzed bar and below the zero level at the previous one.  • **Divergence** — the first analyzed bottom of the oscillator is higher than the

Signal Type	Description of Conditions
	previous one, and the corresponding price bottom is lower than the previous one. 
For selling	• **Reverse** — the oscillator turned downwards (the oscillator falls at the analyzed bar and rises at the previous one).  • **Crossing the zero level** — the main line is below the zero level at the analyzed bar and above the zero level at the previous one.

Signal Type	Description of Conditions
	 • **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

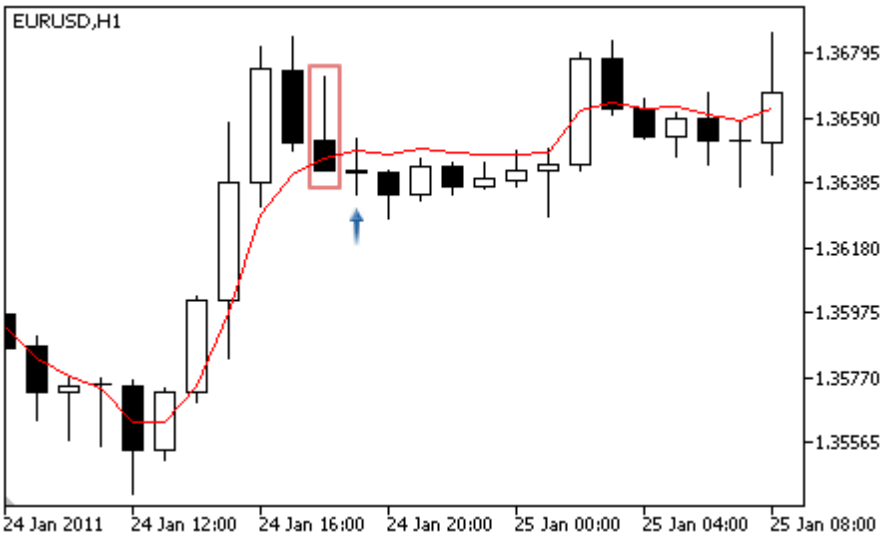
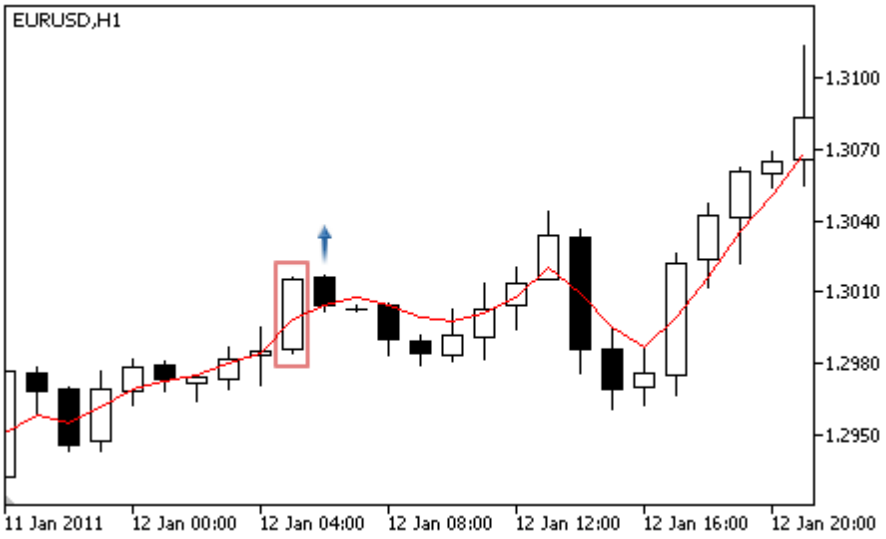
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodTriX	Period of calculation of the oscillator.
Applied	A price series used for calculation of the oscillator.

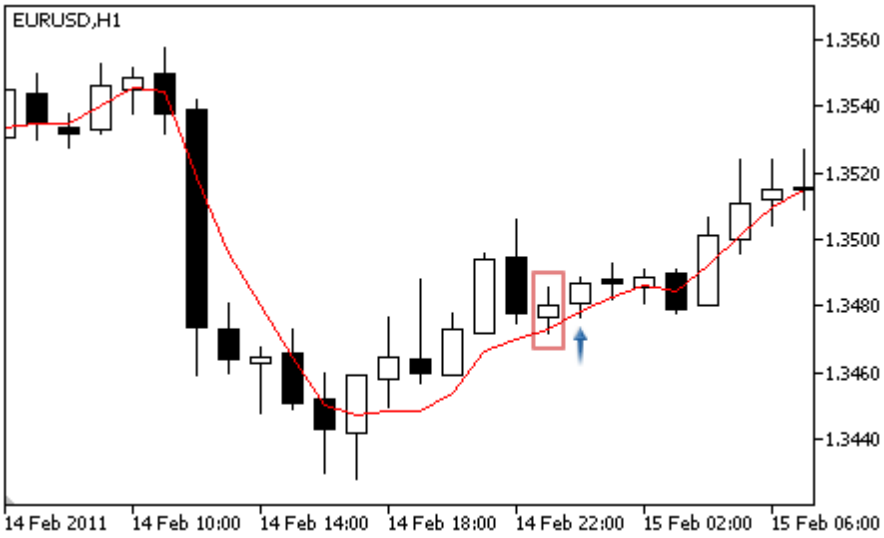
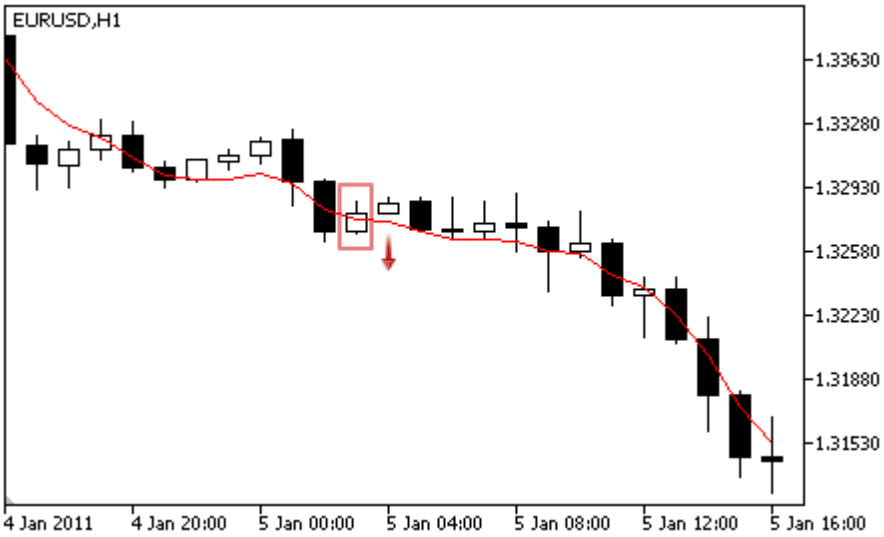
Signals of the Indicator Triple Exponential Moving Average

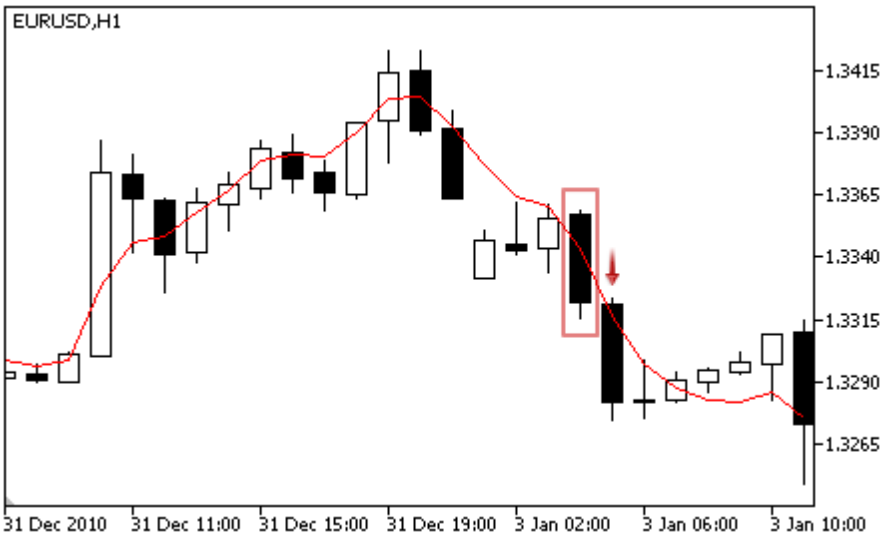
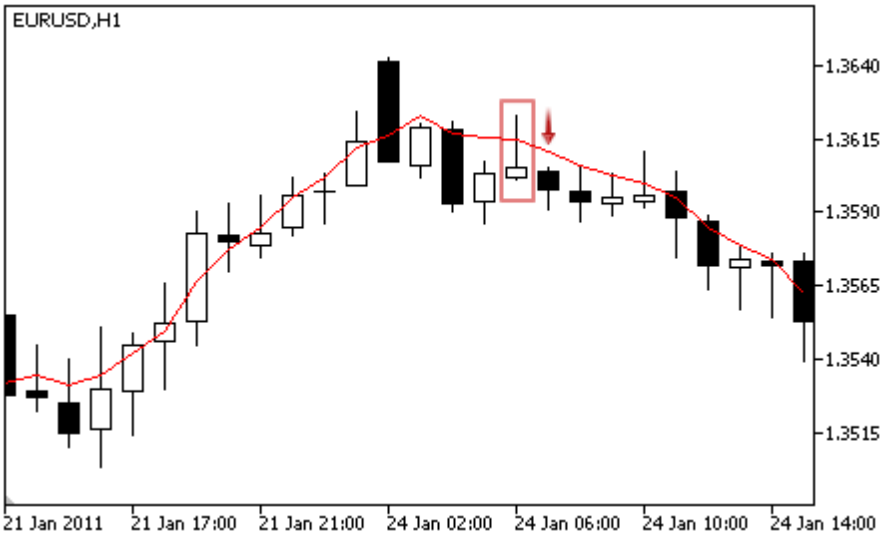
This module of signals is based on the market models of the indicator Triple Exponential Moving Average. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator rises (weak signal).  • **Moving Average crossover.** The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator rises (strong signal). 

Signal Type	Description of Conditions
	• The lower shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is above the indicator, and the Low price is below the indicator) and the indicator rises (weak signal). 
For selling	• The price has crossed the indicator upwards (the Open price of the analyzed bar is below the indicator and the Close price is above the indicator) and the indicator falls (weak signal).  • **Moving Average crossover.** The price has crossed the indicator downwards (the Open price of the analyzed bar is above the indicator and the Close price is below the indicator) and the indicator falls (strong signal).

Signal Type	Description of Conditions
	 • The upper shadow of the bar has crossed the indicator (the Open and Close prices of the analyzed bar is below the indicator, and the High price is above the indicator) and the indicator falls (weak signal). 
No objections to buying	The prices is above the indicator.
No objections to selling	The price is below the indicator.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Adjustable Parameters

This module has the following adjustable parameters:

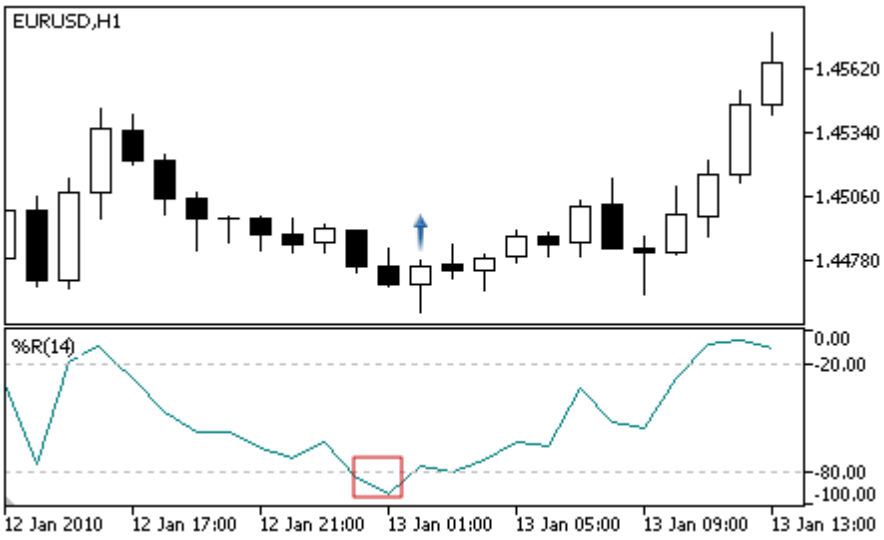
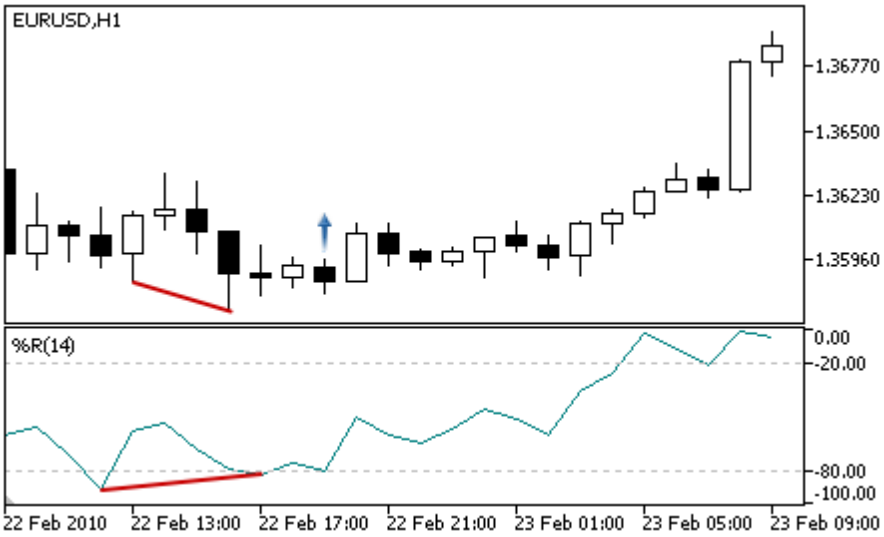
Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodMA	Period of averaging of the indicator.
Shift	Shit of the indicator along the time axis (in bars).
Method	Method of averaging.
Applied	A price series used for calculation of the indicator.

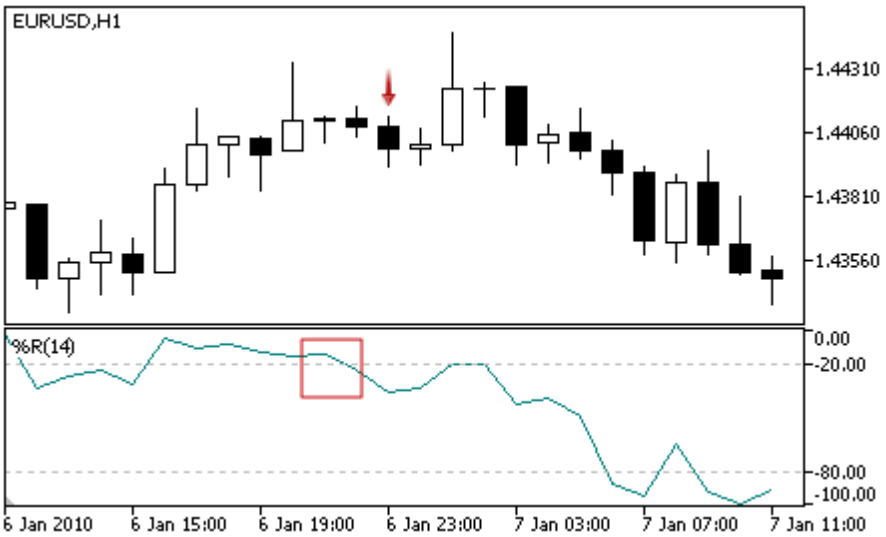
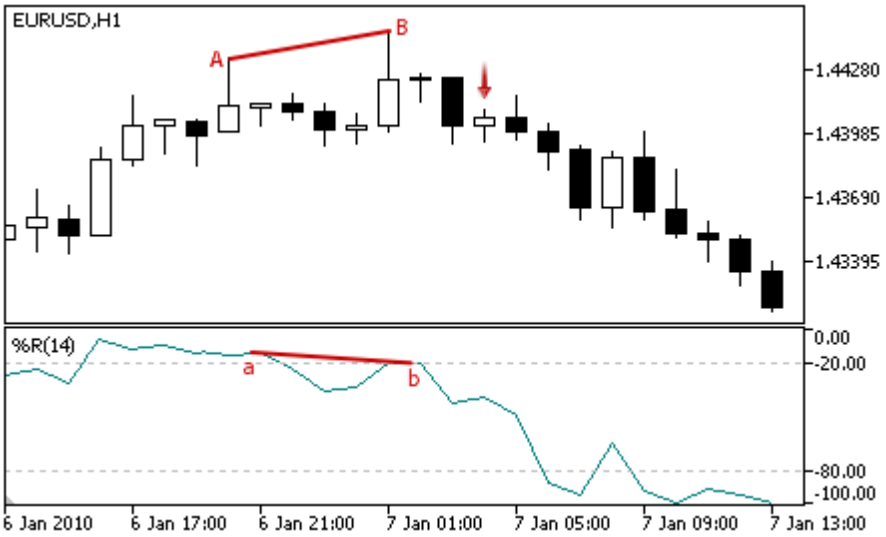
Signals of the Oscillator Williams Percent Range

This module of signals is based on the market models of the oscillator Williams Percent Range. The mechanism of making trade decisions based on signals obtained from the modules is described in a separate section.

Conditions of Generation of Signals

Below you can find the description of conditions when the module passes a signal to an Expert Advisor.

Signal Type	Description of Conditions
For buying	• **Reverse behind the oversold level** — the oscillator turned upwards and its value at the analyzed bar is behind the oversold level (default value is -20).  • **Divergence** — the first analyzed bottom of the oscillator is higher than the previous one, and the corresponding price bottom is lower than the previous one. 
For selling	• **Reverse behind the overbought level** — the oscillator turned downwards and its

Signal Type	Description of Conditions
	value at the analyzed bar is behind the overbought level (default value is -80).  • **Divergence** — the first analyzed peak of the oscillator is lower than the previous one, and the corresponding price peak is higher than the previous peak. 
No objections to buying	Value of the oscillator grows at the analyzed bar.
No objections to selling	Value of the oscillator falls at the analyzed bar.

Note

Depending on the mode of operation of an Expert Advisor ("Every tick" or "Open prices only") an analyzed bar is either the current bar (with index 0), or the last formed bar (with index 1).

Remember that the oscillator Williams Percent Range has a reversed scale. Its maximum value is -

100, and minimum is 0.

Adjustable Parameters

This module has the following adjustable parameters:

Parameter	Description
Weight	Weight of signal of the module in the interval 0 to 1.
PeriodWPR	Period of calculation of the oscillator.

Trailing Stop classes

This section contains technical details of working with trailing stop classes and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and testing) of trading strategies.

MQL5 Standard Library (in terms of trading strategies) is placed in the terminal directory, in the Include\Expert\Trailing folder.

Class	Description
CTrailingFixedPips	This class implements Trailing Stop algorithm, based on fixed points
CTrailingMA	This class implements Trailing Stop algorithm, based on the values of Moving Average indicator
CTrailingNone	A gag class, it doesn't uses any Trailing Stop algorithm
CTrailingPSAR	This class implements Trailing Stop algorithm, based on the values of Parabolic SAR indicator

CTrailingFixedPips

CTrailingFixedPips is a class with implementation of Trailing Stop algorithm, based on fixed points trailing.

If position has Stop Loss price, it checks the minimal allowed Stop Loss distance to the current price. If its value is lower, that Stop Loss level, it suggests to set new Stop Loss price. For this case if position has Take Profit price, it suggests to set new Take Profit price.

If Expert Advisor has been initialized with the flag every_tick=false, it will perform all operations (trading, trailing, etc) only at the new bar. For this case Take profit level can be used. It will allow you to close opened position at Take Profit price before the new bar will be completed.

Description

CTrailingFixedPips implements the Trailing Stop algorithm, based on positions trailing with the fixed points.

Declaration

```
class CTrailingFixedPips: public CExpertTrailing
```

Title

```
   #include <Expert\Trailing\CTrailingFixedPips.mqh>
```

Class Methods

Initialization	
StopLevel	Sets the value of Stop Loss level
ProfitLevel	Sets the value of Take Profit level
virtual ValidationSettings	Checks the settings
Check Trailing Methods	
virtual CheckTrailingStopLong	Check Trailing Stop conditions of long position
virtual CheckTrailingStopShort	Check Trailing Stop conditions of short position

StopLevel

Sets the value Stop Loss level (in points).

```
void  StopLevel(
   int  stop_level      // Stop Loss level
   )
```

Parameters

stop_loss

[in] The value of Stop Loss level (in conventional 2/4-digit points).

Note

If Stop Loss level is equal to 0, the Trailing Stop is not used.

ProfitLevel

Sets the value of Take Profit level (in points).

```
void  ProfitLevel(
   int  profit_level      // Take profit level
   )
```

Parameters

profit_level

[in] The value of Take Profit level (in conventional 2/4-digit points).

Note

If profit level is equal to 0, the Trailing Stop is not used.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Returned value

true if successful, otherwise false.

Note

The function checks Take Profit and Stop Loss levels. The correct values are 0 and values, greater than minimal stop for stop orders for the symbol.

CheckTrailingStopLong

Checks Trailing Stop conditions of long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,     // CPositionInfo object pointer
   double&         sl,           // Stop Loss price
   double&         tp            // Take Profit price
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Variable for Stop Loss price.

tp

[in][out] Variable for Take Profit price.

Returned value

true if conditions are satisfied, otherwise false.

Note

If Stop Loss level is equal to 0, the Trailing Stop is not used. If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price.

If the current Bid price is higher than base price+stop loss level, it suggests to set new Stop Loss price. In this case, If position already has Take Profit price, it suggests to set new Take Profit price equal to Bid price+take proft level.

CheckTrailingStopShort

Checks Trailing Stop conditions of short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,     // CPositionInfo object pointer
   double&         sl,           // Stop Loss price
   double&         tp            // Take Profit price
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Variable for Stop Loss price.

tp

[in][out] Variable for Take Profit price.

Returned value

true if conditions are satisfied, otherwise false.

Note

If Stop Loss level is equal to 0, the Trailing Stop is not used. If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price.

If the current Ask price is lower than base price-stop loss level, it suggests to set new Stop Loss price. In this case, If position already has Take Profit price, it suggests to set new Take Profit price equal to Ask price-take proft level.

CTrailingMA

CTrailingMA is a class with implementation of Trailing Stop algorithm, based on the values of moving average indicator.

Description

CTrailingMA class implements Trailing Stop algorithm, based on the values of moving average indicator of the previous (completed) bar.

Declaration

```
class CTrailingMA: public CExpertTrailing
```

Title

```
   #include <Expert\Trailing\TrailingMA.mqh>
```

Class Methods

Initialization	
Period	Sets period of moving average
Shift	Sets shift of moving average
Method	Sets smoothing method of moving average
Applied	Sets applied price of moving average
virtual InitIndicators	Initializes indicators and time series
virtual ValidationSettings	Checks the settings
Check Trailing Methods	
virtual CheckTrailingStopLong	Check Trailing Stop conditions of long position
virtual CheckTrailingStopShort	Check Trailing Stop conditions of short position

Period

Sets period of moving average.

```
void  Period(
   int  period      // Smoothing period
   )
```

Parameters

period

[in] Period of moving average.

Shift

Sets shift of moving average.

```
void  Shift(
   int  shift      // Shift
   )
```

Parameters

shift

[in] Shift of moving average.

Method

Sets smoothing method of moving average.

```
void  Method(
   ENUM_MA_METHOD  method      // Smoothing method
   )
```

Parameters

method

[in] Smoothing method of moving average indicator.

Applied

Sets applied price of moving average.

```
void  Applied(
   ENUM_APPLIED_PRICE  applied      // Applied price
   )
```

Parameters

applied

[in] Applied price of moving average.

InitIndicators

Initializes indicators and time series.

```
virtual bool  InitIndicators(
   CIndicators*  indicators      // CIndicators collection pointer
   )
```

Parameters

indicators

[in] Pointer to indicators and time series collection (CExpert class member).

Returned value

true if successful, otherwise false.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Returned value

true if successful, otherwise false.

Note

The function checks the period of moving average, the correct values are positive.

CheckTrailingStopLong

Checks Trailing Stop conditions of long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,     // CPositionInfo object pointer
   double&         sl,           // Stop Loss price
   double&         tp            // Take Profit price
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Variable for Stop Loss price.

tp

[in][out] Variable for Take Profit price.

Returned value

true if conditions are satisfied, otherwise false.

Note

The first it calculates the maximal allowed Stop Loss price, closest to the current price and calculates Stop Loss price using the values of moving average indicator of the previous (completed) bar.

If position already has Stop Loss price, its value is assumed as a base price, otherwise the base price is the open price of the position.

If the calculated Stop Loss price is higher than base price and lower than maximal allowed Stop Loss price, it suggests to set new Stop Loss price.

CheckTrailingStopShort

Checks Trailing Stop conditions of short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,     // CPositionInfo object pointer
   double&         sl,           // Stop Loss price
   double&         tp            // Take Profit price
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Variable for Stop Loss price.

tp

[in][out] Variable for Take Profit price.

Returned value

true if conditions are satisfied, otherwise false.

Note

The first it calculates the minimal allowed Stop Loss price, closest to the current price and calculates Stop Loss price using the values of moving average indicator of the previous (completed) bar.

If position already has Stop Loss price, its value is assumed as a base price, otherwise the base price is the open price of the position.

If the calculated Stop Loss price is higher than base price and lower than minimal allowed Stop Loss price, it suggests to set new Stop Loss price.

CTrailingNone

CTrailingNone is a gag class. This class should be used at initialization of Trailng object if your strategy doesn't use Trailing Stop.

Description

CTrailingNone class doesn't implement any Trailing Stop algorithms. The methods of checking Trailing Stop conditions always return false.

Declaration

```
class CTrailingNone: public CExpertTrailing
```

Title

```
   #include <Expert\Trailing\TrailingNone.mqh>
```

Class Methods

Check Trailing Methods	
virtual CheckTrailingStopLong	A gag method for check Trailing Stop conditions of long position
virtual CheckTrailingStopShort	A gag method for check Trailing Stop conditions of short position

CheckTrailingStopLong

Checks Trailing Stop conditions of long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,     // CPositionInfo object pointer
   double&         sl,           // Stop Loss price
   double&         tp            // Take Profit price
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Variable for Stop Loss price.

tp

[in][out] Variable for Take Profit price.

Returned value

true if conditions are satisfied, otherwise false.

Note

The function always returns false.

CheckTrailingStopShort

Checks Trailing Stop conditions of short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,     // CPositionInfo object pointer
   double&         sl,           // Stop Loss price
   double&         tp            // Take Profit price
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Variable for Stop Loss price.

tp

[in][out] Variable for Take Profit price.

Returned value

true if conditions are satisfied, otherwise false.

Note

The function always returns false.

CTrailingPSAR

CTrailingPSAR is a class with implementation of Trailing Stop algorithm, based on the values of of Parabolic SAR indicator.

Description

CTrailingPSAR class implements the Trailing Stop algorithm, based on the values of Parabolic SAR indicator of the previous (completed) bar.

Declaration

```
class CTrailingPSAR: public CExpertTrailing
```

Title

```
   #include <Expert\Trailing\TrailingParabolicSAR.mqh>
```

Class Methods

Initialization	
Step	Sets the value of step of Parabolic SAR indicator
Maximum	Sets the value of maximum of Parabolic SAR indicator
virtual InitIndicators	Initializes indicators and time series
Check Trailing Methods	
virtual CheckTrailingStopLong	Check conditions of trailing stop of long position
virtual CheckTrailingStopShort	Check conditions of trailing stop of short position

Step

Sets the value of step of Parabolic SAR indicator.

```
void  Step(
   double  step      // Step
   )
```

Parameters

step

[in] The value of Step of Parabolic SAR indicator.

Maximum

Sets the value of maximum of Parabolic SAR indicator.

```
void  Maximum(
   double  maximum      // Maximum
   )
```

Parameters

maximum

[in] The value of maximum of Parabolic SAR indicator.

InitIndicators

Initializes indicators and time series.

```
virtual bool  InitIndicators(
   CIndicators*  indicators      // CIndicators collection pointer
   )
```

Parameters

indicators

[in] Pointer to indicators and time series collection (CExpert class member).

Returned value

true if successful, otherwise false.

CheckTrailingStopLong

Checks Trailing Stop conditions of long position.

```
virtual bool  CheckTrailingStopLong(
   CPositionInfo*  position,     // Pointer
   double&         sl,           // Link
   double&         tp            // Link
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Variable for Stop Loss price.

tp

[in][out] Variable for Take Profit price.

Returned value

true if conditions are satisfied, otherwise false.

Note

The first it calculates the maximal allowed Stop Loss price, closest to the current price and calculates Stop Loss price using the values of Parabolic SAR indicator of the previous (completed) bar.

If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price.

If the calculated Stop Loss price is higher than base price and lower than maximal allowed Stop Loss price, it suggests to set new Stop Loss price.

CheckTrailingStopShort

Checks Trailing Stop conditions of short position.

```
virtual bool  CheckTrailingStopShort(
   CPositionInfo*  position,     // Pointer
   double&         sl,           // Link
   double&         tp            // Link
   )
```

Parameters

position

[in] Pointer to CPositionInfo object.

sl

[in][out] Variable for Stop Loss price.

tp

[in][out] Variable for Take Profit price.

Returned value

true if conditions are satisfied, otherwise false.

Note

The first it calculates the minimal allowed Stop Loss price, closest to the current price and calculates Stop Loss price using the values of Parabolic SAR indicator of the previous (completed) bar.

If position already has Stop Loss price, its value is assumed as a base price, otherwise the position open price is assumed as a base price.

If the calculated Stop Loss price is higher than base price and lower than minimal allowed Stop Loss price, it suggests to set new Stop Loss price.

Money Management classes

This section contains technical details of working with money and risk management classes and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating (and testing) of trading strategies.

MQL5 Standard Library (in terms of money and risk management classes) is placed in the terminal directory, in the Include\Expert\Money\ folder.

Class	Description
CMoneyFixedLot	This class implements money management algorithm, based on trading with predefined fixed lot size.
CMoneyFixedMargin	This class implements money management algorithm, based on trading with predefined fixed margin.
CMoneyFixedRisk	This class implements money management algorithm, based on trading with predefined risk.
CMoneyNone	This class implements money management algorithm, based on trading with minimal allowed lot size.
CMoneySizeOptimized	This class implements money management algorithm, based on trading with variable lot size, depending on results of the previous deals.

CMoneyFixedLot

CMoneyFixedLot is the class money management algorithm, based on trading with predefined fixed lot size.

Description

CMoneyFixedLot implements money management algorithm, based on trading with predefined fixed lot size.

Declaration

```
class CMoneyFixedLot: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyFixedLot.mqh>
```

Class Methods

Initialization	
Lots	Sets trading volume
virtual ValidationSettings	Checks the settings
Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for long position
virtual CheckOpenShort	Gets trade volume for short position

Lots

Sets trading volume (in lots).

```
void  Lots(
   double  lots      // Lots
   )
```

Parameters

lots

[in] Trading volume (in lots).

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Returned value

true if successful, otherwise false.

Note

Checks the specified trading volume for correctness.

CheckOpenLong

Gets trade volume for long position.

```
virtual double  CheckOpenLong(
   double  price,     // Price
   double  sl         // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for long position.

Note

The function always returns the fixed trade volume, defined by Lots method.

CheckOpenShort

Gets trade volume for short position.

```
virtual double  CheckOpenShort(
   double  price,     // Price
   double  sl         // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for short position.

Note

The function always returns the fixed trade volume, defined by Lots method.

CMoneyFixedMargin

CMoneyFixedMargin is the class money management algorithm, based on trading with predefined fixed margin.

Description

CMoneyFixedMargin implements money management algorithm, based on trading with predefined fixed margin.

Declaration

```
class CMoneyFixedMargin: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyFixedMargin.mqh>
```

Class Methods

Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for long position
virtual CheckOpenShort	Gets trade volume for short position

CheckOpenLong

Gets trade volume for long position.

```
virtual double  CheckOpenLong(
   double  price,     // Price
   double  sl         // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for long position.

Note

The function returns trade volume for long position, it uses the fixed margin. The margin is defined by Percent parameter of CExpertMoney base class.

CheckOpenShort

Gets trade volume for short position.

```
virtual double  CheckOpenShort(
   double  price,      // Price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for short position.

Note

The function returns trade volume for short position, it uses the fixed margin. The margin is defined by Percent parameter of CExpertMoney base class.

CMoneyFixedRisk

CMoneyFixedRisk is a class with implementation of money management algorithm with fixed predefined risk.

Description

CMoneyFixedRisk class implements the money management algorithm with fixed predefined risk.

Declaration

```
class CMoneyFixedRisk: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyFixedRisk.mqh>
```

Class Methods

Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for long position
virtual CheckOpenShort	Gets trade volume for short position

CheckOpenLong

Gets trade volume for long position.

```
virtual double  CheckOpenLong(
   double  price,      // Price
   double  sl          // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for long position.

Note

The function returns trade volume for long position, it uses the fixed risk. The risk is defined by Percent parameter of CExpertMoney base class.

CheckOpenShort

Gets trade volume for short position.

```
virtual double  CheckOpenShort(
   double  price,     // Price
   double  sl         // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for short position.

Note

The function returns trade volume for short position, it uses the fixed risk. The risk is defined by Percent parameter of CExpertMoney base class.

CMoneyNone

CMoneyNone is a class with implementation of trading algorithm with minimal allowed lot.

Description

CMoneyNone class implements trading with minimal allowed lot.

Declaration

```
class CMoneyNone: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneyNone.mqh>
```

Class Methods

Initialization	
virtual ValidationSettings	Checks the settings
Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for long position
virtual CheckOpenShort	Gets trade volume for short position

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Returned value

true if successful, otherwise false.

Note

The function always returns true.

CheckOpenLong

Gets trade volume for long position.

```
virtual double  CheckOpenLong(
   double  price,     // Price
   double  sl         // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for long position.

Note

The function always returns the minimal lot size.

CheckOpenShort

Gets trade volume for long position.

```
virtual double  CheckOpenShort(
   double  price,     // Price
   double  sl         // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for short position.

Note

The function always returns the minimal lot size.

CMoneySizeOptimized

CMoneySizeOptimized is a class with implementation of money management algorithm, based on trading with variable lot size, depending on results of the previous deals.

Description

CMoneySizeOptimized implements money management algorithm, based on trading with variable lot size, depending on results of the previous deals.

Declaration

```
class CMoneySizeOptimized: public CExpertMoney
```

Title

```
#include <Expert\Money\MoneySizeOptimized.mqh>
```

Class Methods

Initialization	
DecreaseFactor	Sets the value of decrease factor
virtual ValidationSettings	Checks the settings
Money and Risk Management Methods	
virtual CheckOpenLong	Gets trade volume for long position
virtual CheckOpenShort	Gets trade volume for short position

DecreaseFactor

Sets the value of decrease factor.

```
void  DecreaseFactor(
   double  decrease_factor      // Decrease factor
   )
```

Parameters

decrease_factor

[in] Decrease factor.

Note

The DecreaseFactor defines the volume decreasing coefficient (compared with the volume of previous position) for the case of consecutive loss trades.

ValidationSettings

Checks the settings.

```
virtual bool  ValidationSettings()
```

Returned value

true if successful, otherwise false.

Note

If the value of decrease factor is negative, it returns false, otherwise it returns true.

CheckOpenLong

Gets trade volume for long position.

```
virtual double  CheckOpenLong(
   double  price,     // Price
   double  sl         // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for long position.

Note

The function returns trade volume for long position, the volume dependent on results of the previous deals.

CheckOpenShort

Gets trade volume for short position.

```
virtual double  CheckOpenShort(
   double  price,     // Price
   double  sl         // Stop Loss price
   )
```

Parameters

price

[in] Price.

sl

[in] Stop Loss price.

Returned value

Trade volume for long position.

Note

The function returns trade volume for short position, the volume dependent on results of the previous deals.

Classes for Creation of Control Panels and Dialogs

This section contains technical details of working with classes for creation of controls panels and description of the relevant components of the MQL5 standard library.

The use of these classes will save time when creating the control panels for MQL5 programs (Expert Advisors and indicators).

MQL5 Standard Library (in terms of controls) is placed in the client terminal data folder, in the MQL5 \Include\Controls.

The example of Expert Advisor, which illustrates the working of these classes can be found in MQL5 \Expert\Examples\Controls.

Auxiliary classes	Description
CPoint	Class of the point in Cartesian coordinates
CRect	Class of the rectangular area

Base classes	Description
CWnd	Base class for all controls
CWndObj	Base class for controls and dialogs
CWndContainer	Base class for complex controls (containing dependent controls)

Simple controls	Description
CLabel	Control, based on "Text label" graphic object
CBmpButton	Control, based on "Bitmap label" graphic object
CButton	Control, based on "Button" graphic object
CEdit	Control, based on "Edit field" graphic object
CPanel	Control, based on "Rectangle label"
CPicture	Control, based on "Bitmap label"

Complex controls	Description
CScroll	Base class of the scroll bar
CScrollV	Vertical scroll bar
CScrollH	Horizontal scroll bar
CWndClient	Base class of the client area with scroll bars
CListView	ListView
CComboBox	ComboBox

CCheckBox	CheckBox
CCheckGroup	CheckGroup
CRadioButton	RadioButton
CCheckGroup	CheckGroup
CSpinEdit	SpinEdit
CDialog	Dialog
CAppDialog	Application Dialog

CPoint

CPoint is a class of the point.

Description

CPoint is a class of the point in Cartesian coordinates.

Declaration

```
class CPoint
```

Title

```
#include <Controls\Rect.mqh>
```

Class Methods

Geometry	
Move	Sets new coordinates of the point
Shift	Performs the relative shift of the point coordinates
Additional methods	
Format	Gets the point coordinates as string

Move

Sets new coordinates of the point.

```
void  Move(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   )
```

Parameters

x

[in] New X coordinate.

y

[in] New Y coordinate.

Returned value

None.

Shift

Performs the relative shift of the point coordinates.

```
void  Shift(
   const int  dx,      // delta X
   const int  dy       // delta Y
   )
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Returned value

None.

Format

Gets the point coordinates as string.

```
string  Format(
   string &  fmt,     // format
   ) const
```

Parameters

fmt

[in] String with format.

Returned value

String with the point coordinates.

CRect

CRect is a class of the rectangular area of the chart.

Description

CRect is a class of the area, it defined by coordinates of the upper-left and lower-right corners of a rectangle in Cartesian coordinates.

Declaration

```
class CRect
```

Title

```
#include <Controls\Rect.mqh>
```

Class Methods

Properties	
Left	Gets/Sets the X coordinate of the upper-left corner
Top	Gets/Sets the Y coordinate of the upper-left corner
Right	Gets/Sets the X coordinate of the lower-right corner
Bottom	Gets/Sets the Y coordinate of the lower-right corner
Width	Gets/Sets the width
Height	Gets/Sets the height
SetBound	Sets new coordinates of using CRect class
Move	Sets new coordinates of the CRect class
Shift	Performs the relative shift of the CRect coordinates
Contains	Checks if the point is inside the CRect class area
Additional methods	
Format	Gets the area coordinates as string

Left (Get Method)

Gets the X coordinate of the upper-left corner.

```
int  Left()
```

Returned value

X coordinate of the upper-left corner.

Left (Set Method)

Sets the X coordinate of the upper-left corner.

```
void  Left(
   const int  x      // new x coordinate
   )
```

Parameters

x

[in] New X coordinate of the upper-left corner.

Returned value

None.

Top (Get Method)

Gets the Y coordinate of the upper-left corner.

```
int  Top()
```

Returned value

Y coordinate of the upper-left corner.

Top (Set Method)

Sets the Y coordinate of the upper-left corner.

```
void  Top(
   const int  y      // y coordinate
   )
```

Parameters

y

[in] New Y coordinate of the upper-left corner.

Returned value

None.

Right (Get Method)

Gets the X coordinate of the lower-right corner.

```
int  Right()
```

Returned value

X coordinate of the lower-right corner.

Right (Set Method)

Sets the Y coordinate of the lower-right corner.

```
void  Right(
   const int  x      // x coordinate
   )
```

Parameters

x

[in] New X coordinate of the lower-right corner.

Returned value

None.

Bottom (Get Method)

Gets the Y coordinate of the lower-right corner.

```
int  Bottom()
```

Returned value

Y coordinate of the lower-right corner.

Bottom (Set Method)

Sets the Y coordinate of the lower-right corner.

```
void  Bottom(
   const int  y      // y coordinate
   )
```

Parameters

y

[in] New Y coordinate of the lower-right corner.

Returned value

None.

Width (Get Method)

Gets the width of the area.

```
int  Width()
```

Returned value

Width of the area.

Width (Set Method)

Sets new width of the area.

```
virtual bool  Width(
   const int  w      // width
   )
```

Parameters

w

[in] New width.

Returned value

true if successful, otherwise false.

Height (Get Method)

Gets the height of the area.

```
int  Height()
```

Returned value

Height of the area.

Height (Set Method)

Sets new height of the area.

```
virtual bool  Height(
   const int  h      // height
   )
```

Parameters

h

[in] New height.

Returned value

true if successful, otherwise false.

SetBound

Sets new coordinates of the area using CRect class coordinates.

```
void  SetBound(
   const & CRect  rect      // CRect class
   )
```

Returned value

None.

SetBound

Sets new coordinates of the area.

```
void  SetBound(
   const int  l      // left
   const int  t      // topt
   const int  r      // right
   const int  b      // bottom
   )
```

Parameters

l

[in] X coordinate of the upper-left corner.

t

[in] Y coordinate of the upper-left corner.

r

[in] X coordinate of the lower-right corner.

b

[in] Y coordinate of the lower-right corner.

Returned value

None.

Move

Sets new coordinates of the CRect class.

```
void  Move(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   )
```

Parameters

x

[in] New X coordinate.

y

[in] New Y coordinate.

Returned value

None.

Shift

Performs the relative shift of the CRect class coordinates.

```
void  Shift(
   const int  dx,      // delta X
   const int  dy       // delta Y
   )
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Returned value

None.

Contains

Checks if the point is inside the CRect class area.

```
bool  Contains(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   )
```

Parameters

x

[in] X coordinate.

y

[in] Y coordinate.

Returned value

true, if the point is inside the area (including borders), otherwise - false.

Format

Gets the area coordinates as string.

```
string  Format(
   string &  fmt,     // format
   ) const
```

Parameters

fmt

[in] String with format.

Returned value

String with the area coordinates.

CWnd

CWnd is a base class for all controls, included in the MQL5 Standard Library.

Description

CWnd class is the implementation of the base control class.

Declaration

```
class CWnd : public CObject
```

Title

```
#include <Controls\Wnd.mqh>
```

Class Methods

Create and destroy	
Create	Creates control
Destroy	Destroys control
Chart event handlers	
OnEvent	Event handler of all chart events
OnMouseEvent	Event handler for the CHARTEVENT_MOUSE_MOVE event
Name	
Name	Gets control name
Access to container	
ControlsTotal	Gets the number of controls in the container
Control	Gets the control by index
ControlFind	Gets the control by ID
Geometry	
Rect	Gets the pointer to the CRect class object
Left	Gets/Sets the X-coordinate of the upper-left corner
Top	Gets/Sets the Y-coordinate of the upper-left corner
Right	Gets/Sets the X-coordinate of the lower-right corner
Bottom	Gets/Sets the Y-coordinate of the lower-right corner

Width	Gets/Sets the width
Height	Gets/Sets the height
Move	Sets new coordinates of the control
Shift	Performs the relative shift of the control coordinates
Resize	Sets new width/height of the control
Contains	Checks if the point/control is inside the control area
Align	
Alignment	Sets alignment properties of the control
Align	Performs control alignment
Identification	
Id	Gets/Sets the control ID
State	
IsEnabled	Gets a value indicating whether the control is enabled
Enable	Sets a value indicating whether the control is enabled
Disable	Disables the control
IsVisible	Checks the visibility flag
Visible	Sets the visibility flag
Show	Shows the control
Hide	Hides the control
IsActive	Checks the control activity
Activate	Activates the control
Deactivate	Deactivates the control
State flags	
StateFlags	Gets/Sets the control state flags
StateFlagsSet	Sets the control state flags
StateFlagsReset	Resets the control state flags
Properties flags	
PropFlags	Gets/Sets the control properties flags
PropFlagsSet	Sets the control properties flags
PropFlagsReset	Resets the control properties flags

Mouse operations	
MouseX	Gets/Saves the mouse X coordinate
MouseY	Gets/Saves the mouse Y coordinate
MouseFlags	Gets/Saves the mouse buttons state
MouseFocusKill	Kills mouse focus
Internal event handlers	
OnCreate	"Create" event handler
OnDestroy	"Destroy" event handler
OnMove	"Move" event handler
OnResize	"Resize" event handler
OnEnable	"Enable" event handler
OnDisable	"Disable" event handler
OnShow	"Show" event handler
OnHide	"Hide" event handler
OnActivate	"Activate" event handler
OnDeactivate	"Deactivate" event handler
OnClick	"Click" event handler
OnChange	"Change" event handler
Mouse event handlers	
OnMouseDown	"MouseDown" event handler
OnMouseUp	"MouseUp" event handler
Drag event handlers	
OnDragStart	"DragStart" event handler
OnDragProcess	"DragProcess" event handler
OnDragEnd	"DragEnd" event handler
Drag object	
DragObjectCreate	Creates drag object
DragObjectDestroy	Destroys drag object

Create

Creates a control.

```
virtual bool  Create(
   const long     chart,      // chart ID
   const string   name,       // name
   const int      subwin,     // chart subwindow
   const int      x1,         // x1 coordinate
   const int      y1,         // y1 coordinate
   const int      x2,         // x2 coordinate
   const int      y2          // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper-left corner.

y1

[in] Y coordinate of the upper-left corner.

x2

[in] X coordinate of the lower-right corner.

y2

[in] Y coordinate of the lower-right corner.

Returned value

true if successful, otherwise false.

Note

Base class method only saves the parameters and always return true.

Destroy

Destroys a control.

```
virtual bool  Destroy()
```

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

OnMouseEvent

Mouse event handler (the CHARTEVENT_MOUSE_MOVE chart event).

```
virtual bool  OnMouseEvent(
   const int  x,          // x coordinate
   const int  y,          // y coordinate
   const int  flags       // flags
   )
```

Parameters

x

[in] X coordinate of the mouse cursor relative to the upper-left corner of the chart.

y

[in] Y coordinate of the mouse cursor relative to the upper-left corner of the chart.

flags

[in] Flag of mouse buttons states.

Returned value

true - if the event has been processed, otherwise false.

Name

Gets control name.

```
string  Name()  const
```

Returned value

Control name.

ControlsTotal

Gets the number of controls in the container.

```
int  ControlsTotal()  const
```

Returned value

Number of controls in container.

Note

The base class method doesn't have the container, it provides the access to container for its heirs and always returns 0.

Control

Gets the control by index.

```
CWnd*  Control(
   const int  ind      // index
   )  const
```

Parameters

ind

[in] Control index.

Returned value

A pointer to the control.

Note

The base class method doesn't have the container, it provides the access to container for its heirs and always returns NULL.

ControlFind

Gets the control from container by specified ID.

```
virtual CWnd*  ControlFind(
   const long  id      // ID
   )
```

Parameters

id

[in] Identifier of the control to find.

Returned value

Pointer to the control.

Note

The base class method doesn't have the container, it provides the access to container for its heirs. If the specified ID match with the container ID, it returns a pointer to itself (this).

Rect

Gets the pointer to the CRect class object.

```
const CRect*  Rect()  const
```

Returned value

Pointer to the CRect class object.

Left (Get Method)

Gets the X coordinate of the upper-left corner of the control.

```
int  Left()
```

Returned value

X coordinate of the upper-left corner of the control.

Left (Set Method)

Sets the X coordinate of the upper-left corner of the control.

```
void  Left(
   const int  x      // new x coordinate
   )
```

Parameters

x

[in] New X coordinate of the upper-left corner.

Returned value

None.

Top (Get Method)

Gets the Y coordinate of the upper-left corner of the control.

```
int  Top()
```

Returned value

Y coordinate of the upper-left corner of the control.

Top (Set Method)

Sets the Y coordinate of the upper-left corner of the control.

```
void  Top(
   const int  y      // y coordinate
   )
```

Parameters

y

[in] New Y coordinate of the upper-left corner.

Returned value

None.

Right (Get Method)

Gets the X coordinate of the lower-right corner of the control.

```
int  Right()
```

Returned value

X coordinate of the lower-right corner.

Right (Set Method)

Sets the Y coordinate of the lower-right corner of the control.

```
void  Right(
   const int  x      // x coordinate
   )
```

Parameters

x

[in] New X coordinate of the lower-right corner.

Returned value

None.

Bottom (Get Method)

Gets the Y coordinate of the lower-right corner of the control.

```
int  Bottom()
```

Returned value

Y coordinate of the lower-right corner of the control.

Bottom (Set Method)

Sets the Y coordinate of the lower-right corner of the control.

```
void  Bottom(
   const int  y      // y coordinate
   )
```

Parameters

y

[in] New Y coordinate of the lower-right corner.

Returned value

None.

Width (Get Method)

Gets the control width.

```
int  Width()
```

Returned value

Width of the control.

Width (Set Method)

Sets new width of the control.

```
virtual bool  Width(
   const int  w      // width
   )
```

Parameters

w

[in] New width.

Returned value

true if successful, otherwise false.

Height (Get Method)

Gets the control height.

```
int  Height()
```

Returned value

Height of the control.

Height (Set Method)

Sets new height of the control.

```
virtual bool  Height(
   const int  h      // height
   )
```

Parameters

h

[in] New height.

Returned value

true if successful, otherwise false.

Move

Sets new coordinates of the control.

```
void  Move(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   )
```

Parameters

x

[in] New X coordinate.

y

[in] New Y coordinate.

Returned value

None.

Shift

Performs the relative shift of the control coordinates.

```
void  Shift(
   const int  dx,      // delta X
   const int  dy       // delta Y
   )
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Returned value

None.

Resize

Sets new width/height of the control.

```
virtual bool  Resize(
   const int  w,      // width
   const int  h       // height
   )
```

Parameters

w

[in] New width.

h

[in] New height.

Returned value

true if successful, otherwise false.

Contains

Checks if the point is inside the control area of the chart.

```
bool  Contains(
   const int  x,      // X coordinate
   const int  y       // Y coordinate
   )
```

Parameters

x

[in] X coordinate.

y

[in] Y coordinate.

Returned value

true, if the point is inside the area (including borders), otherwise - false.

Contains

Checks if the specified control is inside the control area of the chart.

```
bool  Contains(
   const CWnd*  control       // pointer
   )  const
```

Parameters

control

[in] Object pointer.

Returned value

true, if the specified control is inside the area (including borders), otherwise - false.

Alignment

Sets alignment parameters of the control.

```
void  Alignment(
   const int  flags,       // alignment flags
   const int  left,        // offset from the left border
   const int  top,         // offset from the top border
   const int  right,       // offset from the right border
   const int  bottom       // offset from the bottom border
   )
```

Parameters

flags

[in] Alignment flags.

left

[in] Fixed offset from the left border.

top

[in] Fixed offset from the top border.

right

[in] Fixed offset from the right border.

bottom

[in] Fixed offset from the bottom border.

Returned value

None.

Note

Alignement flags:

```
enum WND_ALIGN_FLAGS
{
   WND_ALIGN_NONE=0,                                  // no align
   WND_ALIGN_LEFT=1,                                  // align left
   WND_ALIGN_TOP=2,                                   // align top
   WND_ALIGN_RIGHT=4,                                 // align right
   WND_ALIGN_BOTTOM=8,                                // align bottom
   WND_ALIGN_WIDTH = WND_ALIGN_LEFT|WND_ALIGN_RIGHT,  // align width
   WND_ALIGN_HEIGHT=WND_ALIGN_TOP|WND_ALIGN_BOTTOM,   // align height
   WND_ALIGN_CLIENT=WND_ALIGN_WIDTH|WND_ALIGN_HEIGHT, // align height and width
   }
```

Align

Performs control alignment in the specified chart area.

```
virtual bool  Align(
   const CRect*  rect      // pointer
   )
```

Parameters

rect

[in] Pointer to the object with chart area coordinates.

Returned value

true if successful, otherwise false.

Note

The alignment parameters must be specified (no alignment by default).

Id (Get Method)

Gets the control ID.

```
long  Id()  const
```

Returned value

The control identifier.

Id (Set Method)

Sets new value of the control ID.

```
virtual long  Id(
   const long  id      // identifier
   )
```

Parameters

id

[in] New value of the control identifier.

Returned value

None.

IsEnabled

Gets a value indicating whether the control is enabled.

```
bool  IsEnabled()  const
```

Returned value

true - if the control is enabled, otherwise - false.

Enable

Enables the control.

```
virtual bool  Enable()
```

Returned value

true if successful, otherwise false.

Note

If the control is enabled, it's able to process the external events.

Disable

Disables the control.

```
virtual bool  Disable()
```

Returned value

true if successful, otherwise false.

Note

The disabled control doesn't able to process the external events.

IsVisible

Gets a value indicating whether the control is visible.

```
bool  IsVisible()  const
```

Returned value

true if the control is shown on the chart, otherwise false.

Visible

Sets the visibility flag.

```
virtual bool  Visible(
   const bool  flag      // flag
   )
```

Parameters

flag

[in] New flag.

Returned value

true if successful, otherwise false.

Show

Shows the control.

```
virtual bool  Show()
```

Returned value

true if successful, otherwise false.

Hide

Hides the control.

```
virtual bool  Hide()
```

Returned value

true if successful, otherwise false.

IsActive

Gets a value indicating whether the control is active.

```
bool  IsActive()  const
```

Returned value

true if the control is active, otherwise false.

Activate

Activates the control.

```
virtual bool  Activate()
```

Returned value

true if successful, otherwise false.

Note

The control becomes active when the mouse cursor is hovering over it.

Deactivate

Deactivates the control.

```
virtual bool  Deactivate()
```

Returned value

true if successful, otherwise false.

Note

The control becomes inactive when the mouse cursor is out off the control.

StateFlags (Get Method)

Gets the control state flags.

```
int  StateFlags()
```

Returned value

The control state flags.

StateFlags (Set Method)

Sets the control state flags.

```
virtual void  StateFlags(
   const int  flags      // flags
   )
```

Parameters

flags

[in] New control state flags.

Returned value

None.

StateFlagsSet

Sets the control state flags.

```
virtual void  StateFlagsSet(
   const int  flags      // flags
   )
```

Parameters

flags

[in] Flags to set (bit mask).

Returned value

None.

StateFlagsReset

Resets the control state flags.

```
virtual void  StateFlagsReset(
   const int  flags      // flags
   )
```

Parameters

flags

[in] Flags to reset (bit mask).

Returned value

None.

PropFlags (Get Method)

Gets the control properties flags.

```
void  PropFlags(
   const int  flags      // flags
   )
```

Returned value

The control properties flags.

PropFlags (Set Method)

Sets the control properties flags.

```
virtual void  PropFlags(
   const int  flags      // flags
   )
```

Parameters

flags

[in] New flags.

Returned value

None.

PropFlagsSet

Sets the control properties flags.

```
virtual void  PropFlagsSet(
   const int  flags      // flags
   )
```

Parameters

flags

[in] Flags to set (bit mask).

Returned value

None.

PropFlagsReset

Resets the control properties flags.

```
virtual void  PropFlagsReset(
   const int  flags      // flags
   )
```

Parameters

flags

[in] Flags to reset (bit mask).

Returned value

None.

MouseX (Set Method)

Saves the mouse X coordinate.

```
void  MouseX(
   const int  value      // coordinate
   )
```

Parameters

value

[in] The X coordinate of the mouse.

Returned value

None.

MouseX (Get Method)

Gets the saved X coordinate of the mouse.

```
int  MouseX()
```

Returned value

The saved X coordinate of the mouse.

MouseY (Set Method)

Saves the mouse Y coordinate.

```
void  MouseY(
   const int  value      // coordinate
   )
```

Parameters

value

[in] The Y coordinate of the mouse.

Returned value

None.

MouseY (Get Method)

Gets the saved Y coordinate of the mouse.

```
int  MouseY()
```

Returned value

The saved Y coordinate of the mouse.

MouseFlags (Set Method)

Saves the state of mouse buttons.

```
virtual void  MouseFlags(
   const int  value      // state
   )
```

Parameters

value

[in] State of mouse buttons.

Returned value

None.

MouseFlags (Get Method)

Gets the saved state of mouse buttons.

```
int  MouseFlags()
```

Returned value

State of mouse buttons.

MouseFocusKill

Clears the saved state of mouse buttons and deactivates the control.

```
bool  MouseFocusKill(
   const long  id=CONTROLS_INVALID_ID      // id
   )
```

Parameters

id=CONTROLS_INVALID_ID

[in] Identifier of the control, that received mouse focus.

Returned value

The control deactivation result.

OnCreate

The control "Create" event handler.

```
virtual bool  OnCreate()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnDestroy

The control "Destroy" event handler.

```
virtual bool  OnDestroy()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnMove

The control "Move" event handler.

```
virtual bool  OnMove()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnResize

The control "Resize" event handler.

```
virtual bool  OnResize()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnEnable

The control "Enable" (if enabled, it can respond to user interaction) event handler.

```
virtual bool  OnEnable()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnDisable

The control "Disable" (if disabled it can't respond to user interaction) event handler.

```
virtual bool  OnDisable()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnShow

The control "Show" event handler.

```
virtual bool  OnShow()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnHide

The control "Hide" event handler.

```
virtual bool  OnHide()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnActivate

The control "Activate" event handler.

```
virtual bool  OnActivate()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnDeactivate

The control "Deactivate" event handler.

```
virtual bool  OnDeactivate()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnClick

The control "Click" (left mouse button click) event handler.

```
virtual bool  OnClick()
```

Returned value

true if the event has been processed, otherwise false.

OnChange

The control "Change" event handler.

```
virtual bool  OnChange()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnMouseDown

The control "MouseDown" event handler.

```
virtual bool  OnMouseDown()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "MouseDown" event occurs when left mouse button is pressed on the control.

OnMouseUp

The control "MouseUp" (left mouse button release) event handler.

```
virtual bool  OnMouseUp()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "MouseUp" event occurs when left mouse button is released on the control.

OnDragStart

The control "DragStart" event handler.

```
virtual bool  OnDragStart()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "DragStart" event occurs at the start of drag operation.

OnDragProcess

The control "DragProcess" event handler.

```
virtual bool  OnDragProcess(
   const int  x,      // x coordinate
   const int  y       // y coordinate
   )
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Returned value

true if the event has been processed, otherwise false.

Note

The "DragProcess" event occurs when the control is moved.

OnDragEnd

The control "DragEnd" event handler.

```
virtual bool  OnDragEnd()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "DragEnd" event occurs when control drag process is finished.

DragObjectCreate

Creates drag object.

```
virtual bool  DragObjectCreate()
```

Returned value

true if successful, otherwise false.

Note

true if the event has been processed, otherwise false.

DragObjectDestroy

Destroys drag object.

```
virtual bool  DragObjectDestroy()
```

Returned value

true if successful, otherwise false.

CWndObj

CWndObj is a base class for simple controls (based on chart objects) of the Standard library.

Description

CWndObj class implements base methods of the simple control.

Declaration

```
class CWndObj : public CWnd
```

Title

```
#include <Controls\WndObj.mqh>
```

Class Methods

Chart events processing	
OnEvent	Event handler of all chart events
Properties	
Text	Gets/Sets the OBJPROP_TEXT property of the chart object
Color	Gets/Sets the OBJPROP_COLOR property of the chart object
ColorBackground	Gets/Sets the OBJPROP_BGCOLOR property of the chart object
ColorBorder	Gets/Sets the OBJPROP_BORDER_COLOR property of the chart object
Font	Gets/Sets the OBJPROP_FONT property of the chart object
FontSize	Gets/Sets the OBJPROP_FONTSIZE property of the chart object
ZOrder	Gets/Sets the OBJPROP_ZORDER property of the chart object
Chart objects event handlers	
OnObjectCreate	CHARTEVENT_OBJECT_CREATE event handler
OnObjectChange	CHARTEVENT_OBJECT_CHANGE event handler
OnObjectDelete	CHARTEVENT_OBJECT_DELETE event handler
OnObjectDrag	CHARTEVENT_OBJECT_DRAG event handler
Properties change event handlers	

OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
OnSetZOrder	"SetZOrder" event handler
Internal event handlers	
OnDestroy	"Destroy" event handler
OnChange	"Change" event handler

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

Text (Get method)

Gets the OBJPROP_TEXT (text) property of the chart object.

```
string  Text()
```

Returned value

The value of the OBJPROP_TEXT property.

Text (Set method)

Sets the OBJPROP_TEXT (text) property of the chart object.

```
bool  Text(
   const string  value      // new value
   )
```

Parameters

value

[in] New value of the OBJPROP_TEXT property.

Returned value

true if successful, otherwise false.

Color (Get method)

Gets the OBJPROP_COLOR (color) property of the chart object.

```
color  Color()
```

Returned value

The value of the OBJPROP_COLOR property.

Color (Set method)

Sets the OBJPROP_COLOR (color) propery of the chart object.

```
bool  Color(
   const color  value      // value
   )
```

Parameters

value

[in] New value of the OBJPROP_COLOR property.

Returned value

true if successful, otherwise false.

ColorBackground (Get method)

Gets the OBJPROP_BGCOLOR (background color) of the chart object.

```
color  ColorBackground()
```

Returned value

The value of the OBJPROP_BGCOLOR property.

ColorBackground (Set method)

Sets the OBJPROP_BGCOLOR (background color) property of the chart object.

```
bool  ColorBackground(
   const color  value      // value
   )
```

Parameters

value

[in] New value of the OBJPROP_BGCOLOR property.

Returned value

true if successful, otherwise false.

ColorBorder (Get method)

Gets the OBJPROP_BORDER_COLOR (border color) property of the chart object.

```
color  ColorBorder()
```

Returned value

The value of the OBJPROP_BORDER_COLOR property.

ColorBorder (Set method)

Sets the OBJPROP_BORDER_COLOR (border color) property of the chart object.

```
bool  ColorBorder(
   const color  value      // value
   )
```

Parameters

value

[in] New value of the OBJPROP_BORDER_COLOR property.

Returned value

true if successful, otherwise false.

Font (Get method)

Gets the OBJPROP_FONT (font) property of the chart object.

```
string  Font()
```

Returned value

The value of the OBJPROP_FONT property.

Font (Set method)

Sets the OBJPROP_FONT (font) property of the chart object.

```
bool  Font(
   const string  value      // new value
   )
```

Parameters

value

[in] New value of the OBJPROP_FONT property.

Returned value

true if successful, otherwise false.

FontSize (Get method)

Gets the OBJPROP_FONTSIZE (font size) property of the chart object.

```
int  FontSize()
```

Returned value

The value of the OBJPROP_FONTSIZE property.

FontSize (Set method)

Sets the OBJPROP_FONTSIZE (font size) property of the chart object.

```
bool  FontSize(
   const int  value      // new value
   )
```

Parameters

value

[in] New value of the OBJPROP_FONTSIZE property.

Returned value

true if successful, otherwise false.

ZOrder (Get method)

Gets the OBJPROP_ZORDER property of the chart object.

```
long  ZOrder()
```

Returned value

The value of the OBJPROP_ZORDER property.

ZOrder (Set method)

Sets the OBJPROP_ZORDER property of the chart object.

```
bool  ZOrder(
   const long  value      // new value
   )
```

Parameters

value

[in] New value of the OBJPROP_ZORDER property.

Returned value

true if successful, otherwise false.

OnObjectCreate

The chart object CHARTEVENT_OBJECT_CREATE event handler.

```
virtual bool  OnObjectCreate()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnObjectChange

The chart object CHARTEVENT_OBJECT_CHANGE event handler.

```
virtual bool  OnObjectChange()
```

Returned value

true if the event has been processed, otherwise false.

OnObjectDelete

The chart object CHARTEVENT_OBJECT_DELETE event handler.

```
virtual bool  OnObjectDelete()
```

Returned value

true if the event has been processed, otherwise false.

OnObjectDrag

The chart object CHARTEVENT_OBJECT_DRAG event handler.

```
virtual bool  OnObjectDrag()
```

Returned value

true if the event has been processed, otherwise false.

OnSetText

The control "SetText" (change of the OBJPROP_TEXT property) event handler.

```
virtual bool  OnSetText()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnSetColor

The control "SetColor" (change of the OBJPROP_COLOR property) event handler.

```
virtual bool  OnSetColor()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnSetColorBackground

The control "SetColorBackground" (change of the OBJPROP_BGCOLOR property) event handler.

```
virtual bool  OnSetColorBackground()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnSetFont

The control "SetFont" (change of the OBJPROP_FONT property) event handler.

```
virtual bool  OnSetFont()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnSetFontSize

The control "SetFontSize" (change of the OBJPROP_FONTSIZE property) event handler.

```
virtual bool  OnSetFontSize()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnSetZOrder

The control "SetZOrder" (change of the OBJPROP_ZORDER property) event handler.

```
virtual bool  OnSetZOrder()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnDestroy

The control "Destroy" event handler.

```
virtual bool  OnDestroy()
```

Returned value

true if the event has been processed, otherwise false.

OnChange

The control "Change" event handler.

```
virtual bool  OnChange()
```

Returned value

true if the event has been processed, otherwise false.

CWndContainer

CWndContainer is a base class for a complex control (containing dependent controls) of the Standard library.

Description

CWndContainer class implements base methods of the complex control.

Declaration

```
class CWndContainer : public CWnd
```

Title

```
#include <Controls\WndContainer.mqh>
```

Class Methods

Destroy	
Destroy	Destroys all the container controls
Chart event handlers	
OnEvent	Event handler of all chart events
OnMouseEvent	The CHARTEVENT_MOUSE_MOVE event handler
Access to container	
ControlsTotal	Gets the number controls in the container
Control	Gets control by index
ControlFind	Gets control by ID
Add/Delete	
Add	Adds control to container
Delete	Deletes control from container
Geometry	
Move	Sets new coordinates for all controls of the container
Shift	Performs the relative shift of the coordinates for all controls of the container
Identification	
Id	Sets the ID for all controls of the container
State	
Enable	Enables all controls of the container

Disable	Disables all controls of the container
Show	Shows all controls of the container
Hide	Hides all controls of the container
Mouse operations	
MouseFocusKill	Kills mouse focus
File operations	
Save	Saves container information to file
Load	Loads container information from file
Internal event handlers	
OnResize	"Resize" event handler
OnActivate	"Activate" event handler
OnDeactivate	"Deactivate" event handler

Destroy

Destroys all the container controls.

```
virtual bool  Destroy()
```

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

OnMouseEvent

Mouse event handler.

```
virtual bool  OnMouseEvent(
   const int  x,          // x coordinate
   const int  y,          // y coordinate
   const int  flags       // flags
   )
```

Parameters

x

[in] X coordinate of the mouse cursor relative to the upper-left corner of the chart.

y

[in] Y coordinate of the mouse cursor relative to the upper-left corner of the chart.

flags

[in] Flag of mouse buttons states.

Returned value

true - if the event has been processed, otherwise false.

ControlsTotal

Gets the number of controls in the container.

```
int  ControlsTotal()  const
```

Returned value

Number of controls in the contrainer.

Control

Gets control from the container by index.

```
CWnd*  Control(
   const int  ind      // index
   )  const
```

Parameters

ind

[in] Index of the control needed.

Returned value

Pointer to the control, otherwise NULL if the control isn't found.

ControlFind

Gets control from the container by identifier.

```
virtual CWnd*  ControlFind(
   const long  id      // id
   )
```

Parameters

id

[in] Control ID.

Returned value

Pointer to the control, otherwise NULL if the control isn't found.

Add

Adds a control to the container.

```
bool  Add(
   CWnd&  control      // reference
   )
```

Parameters

control

[in] Control to add, passed by reference.

Returned value

true if successful, otherwise false.

Delete

Deletes control from the container.

```
bool  Delete(
   CWnd&  control      // reference
   )
```

Parameters

control

[in] Control to delete, passed by reference.

Returned value

true if successful, otherwise false.

Move

Sets new coordinates for all controls of the container.

```
virtual bool  Move(
   const int  x,      // x coordinate
   const int  y       // y coordinate
   )
```

Parameters

x

[in] New X coordinate of the upper-left corner.

y

[in] New Y coordinate of the upper-left coner.

Returned value

true if successful, otherwise false.

Shift

Performs the relative shift of the coordinates for all controls of the container.

```
virtual bool  Shift(
   const int  dx,      // delta x
   const int  dy       // delta y
   )
```

Parameters

dx

[in] Delta X.

dy

[in] Delta Y.

Returned value

true if successful, otherwise false.

Id

Sets the ID for all controls of the container.

```
virtual long  Id(
   const long  id      // identifier
   )
```

Parameters

id

[in] Base group identifier.

Returned value

Number of identifiers, used by container controls.

Enable

Enables all the controls of the container.

```
virtual bool  Enable()
```

Returned value

true if successful, otherwise false.

Disable

Disables all controls of the container.

```
virtual bool  Disable()
```

Returned value

true if successful, otherwise false.

Show

Shows all controls of the container.

```
virtual bool  Show()
```

Returned value

true if successful, otherwise false.

Hide

Hides all controls of the container.

```
virtual bool  Hide()
```

Returned value

true if successful, otherwise false.

MouseFocusKill

Clears the saved state of mouse buttons and deactivates alll controls in the container.

```
bool  MouseFocusKill(
   const long  id=CONTROLS_INVALID_ID      // id
   )
```

Parameters

id=CONTROLS_INVALID_ID

[in] Identifier of the control, that received mouse focus.

Returned value

The controls deactivation result.

Save

Saves container information to file.

```
virtual bool  Save(
   const int  file_handle      // handle
   )
```

Parameters

file_handle

[in] Handle of the binary file (must be opened for writing).

Returned value

true if successful, otherwise false.

Load

Loads container information to file

```
virtual bool  Load(
   const int  file_handle      // handle
   )
```

Parameters

file_handle

[in] Handle of the binary file (must be opened for reading).

Returned value

true if successful, otherwise false.

OnResize

The control "Resize" event handler.

```
virtual bool  OnResize()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnActivate

The control "Activate" event handler.

```
virtual bool  OnActivate()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnDeactivate

The control "Deactivate" event handler.

```
virtual bool  OnDeactivate()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

CLabel

CLabel is class of the simple control, based on "Text label" chart object.

Description

CLabel is intended for creation of simple text labels.

Declaration

```
class CLabel : public CWndObj
```

Title

```
#include <Controls\Label.mqh>
```

Class Methods

Create	
Create	Creates control
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
Internal event handlers	
OnCreate	"Create" event handler
OnShow	"Show" event handler
OnHide	"Hide" event handler
OnMove	"Move" event handler

Create

Creates new CLabel control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnSetText

The control "SetText" (change of the OBJPROP_TEXT property) event handler.

```
virtual bool  OnSetText()
```

Returned value

true if the event has been processed, otherwise false.

OnSetColor

The control "SetColor" (change of the OBJPROP_COLOR property) event handler.

```
virtual bool  OnSetColor()
```

Returned value

true if the event has been processed, otherwise false.

OnSetFont

The control "SetFont" (change of the OBJPROP_FONT property) event handler.

```
virtual bool  OnSetFont()
```

Returned value

true if the event has been processed, otherwise false.

OnSetFontSize

The control "SetFontSize" (change of the OBJPROP_FONTSIZE property) event handler.

```
virtual bool  OnSetFontSize()
```

Returned value

true if the event has been processed, otherwise false.

OnCreate

The control "Create" event handler.

```
virtual bool  OnCreate()
```

Returned value

true if the event has been processed, otherwise false.

OnShow

The control "Show" event handler.

```
virtual bool  OnShow()
```

Returned value

true if the event has been processed, otherwise false.

OnHide

The control "Hide" event handler.

```
virtual bool  OnHide()
```

Returned value

true if the event has been processed, otherwise false.

OnMove

The control "Move" event handler.

```
virtual bool  OnMove()
```

Returned value

true if the event has been processed, otherwise false.

CBmpButton

CBmpButton is class of the simple control, based on "Bitmap label" chart object.

Description

CBmpButton is intended for creation of buttons with graphic image.

Declaration

```
class CBmpButton : public CWndObj
```

Title

```
#include <Controls\BmpButton.mqh>
```

Class Methods

Create	
Create	Creates control
Properties	
Border	Gets/Sets the "Border" property of the control
BmpNames	Sets the name of bmp files of the control
BmpOffName	Gets/Sets the name of bmp file for the OFF state
BmpOnName	Gets/Sets the name of bmp file for the ON state
BmpPassiveName	Gets/Sets the name of bmp file for the passive state
BmpActiveName	Gets/Sets the name of bmp file for the active state
State	
Pressed	Gets/Sets the state of the control
Locking	Gets/Sets the "Locking" property of the control
Internal event handlers	
OnSetZOrder	"SetZOrder" event handler
OnCreate	"Create" event handler
OnShow	"Show" event handler
OnHide	"Hide" event handler
OnMove	"Move" event handler

OnChange	"Change" event handler
OnActivate	"Activate" event handler
OnDeactivate	"Deactivate" event handler
OnMouseDown	"MouseDown" event handler
OnMouseUp	"MouseUp" event handler

Create

Creates new CBmpButton control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

Border (Get method)

Gets the "Border" (border width) property of the control.

```
int  Border()  const
```

Returned value

The "Border" property.

Border (Set method)

Sets the "Border" (border width) property of the control.

```
bool  Border(
   const int  value      // new value
   )
```

Parameters

value

[in] New value of the "Border" property.

Returned value

true if successful, otherwise false.

BmpNames

Sets the name of bmp files of the control

```
bool  BmpNames(
   const string  off="",      // file name
   const string  on=""        // file name
   )
```

Parameters

off=""

[in] Name of bmp file for OFF state.

on=""

[in] Name of bmp file for ON state.

Returned value

true if successful, otherwise false.

BmpOffName (Get method)

Gets the name of bmp file for OFF state.

```
string  BmpOffName()  const
```

Returned value

Name of bmp file for OFF state.

BmpOffName (Set method)

Sets the name of bmp file for OFF state.

```
bool  BmpOffName(
   const string  name      // file name
   )
```

Parameters

name

[in] Name of bmp file for OFF state.

Returned value

true if successful, otherwise false.

BmpOnName (Get method)

Gets the name of bmp file for ON state.

```
string  BmpOnName()  const
```

Returned value

Name of bmp file for ON state.

BmpOnName (Set method)

Sets the name of bmp file for ON state.

```
bool  BmpOnName(
   const string  name      // file name
   )
```

Parameters

name

[in] Name of bmp file for ON state.

Returned value

true if successful, otherwise false.

BmpPassiveName (Get method)

Gets the name of bmp file for the control passive state.

```
string  BmpPassiveName()  const
```

Returned value

Name of bmp file for the control passive state.

BmpPassiveName (Set method)

Sets the name of bmp file for the passive state.

```
bool  BmpPassiveName(
   const string  name      // file name
   )
```

Parameters

name

[in] Name of bmp file for the control passive state.

Returned value

true if successful, otherwise false.

BmpActiveName (Get method)

Gets the name of bmp file for the active state.

```
string  BmpActiveName()  const
```

Returned value

Name of bmp file for the active state.

Note

The control becomes active when the mouse cursor is hovering over it.

BmpActiveName (Set method)

Sets the name of bmp file for the active state.

```
bool  BmpActiveName(
   const string  name      // file name
   )
```

Parameters

name

[in] Name of bmp file for the active state.

Returned value

true if successful, otherwise false.

Pressed (Get method)

Gets the state ("Pressed" property) of the control.

```
bool  Pressed()  const
```

Returned value

Control state.

Pressed (Set method)

Sets the state ("Pressed" property) of the control.

```
bool  Pressed(
   const bool  pressed      // new state
   )
```

Parameters

pressed

[in] New control state.

Returned value

true if successful, otherwise false.

Locking (Get method)

Gets the "Locking" property of the control.

```
bool  Locking()  const
```

Returned value

The value of "Locking" property.

Locking (Set method)

Sets new value of the "Locking" property of the control.

```
void  Locking(
   const bool  locking      // new value
   )
```

Parameters

locking

[in] New value of "Locking" property.

Returned value

None.

OnSetZOrder

The control "SetZOrder" (change of the OBJPROP_ZORDER property) event handler.

```
virtual bool  OnSetZOrder()
```

Returned value

true if the event has been processed, otherwise false.

OnCreate

The control "Create" event handler.

```
virtual bool  OnCreate()
```

Returned value

true if the event has been processed, otherwise false.

OnShow

The control "Show" event handler.

```
virtual bool  OnShow()
```

Returned value

true if the event has been processed, otherwise false.

OnHide

The control "Hide" event handler.

```
virtual bool  OnHide()
```

Returned value

true if the event has been processed, otherwise false.

OnMove

The control "Move" event handler.

```
virtual bool  OnMove()
```

Returned value

true if the event has been processed, otherwise false.

OnChange

The control "Change" event handler.

```
virtual bool  OnChange()
```

Returned value

true if the event has been processed, otherwise false.

OnActivate

The control "Activate" event handler.

```
virtual bool  OnActivate()
```

Returned value

true if the event has been processed, otherwise false.

OnDeactivate

The control "Deactivate" event handler.

```
virtual bool  OnDeactivate()
```

Returned value

true if the event has been processed, otherwise false.

OnMouseDown

The control "MouseDown" event handler.

```
virtual bool  OnMouseDown()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "MouseDown" event occurs when left mouse button is pressed on the control.

OnMouseUp

The control "MouseUp" (left mouse button release) event handler.

```
virtual bool  OnMouseUp()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "MouseUp" event occurs when left mouse button is released on the control.

CButton

CButton is class of the simple control, based on "Button" chart object.

Description

CButton class in intended for creation of simple buttons.

Declaration

```
class CButton : public CWndObj
```

Title

```
#include <Controls\Button.mqh>
```

Class Methods

Create	
Create	Creates control
State	
Pressed	Gets/Sets the "Pressed" property
Locking	Gets/Sets the "Locking" property
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetColorBorder	"SetColorBorder" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
Internal event handlers	
OnCreate	"Create" event handler
OnShow	"Show" event handler
OnHide	"Hide" event handler
OnMove	"Move" event handler
OnResize	"Resize" event handler
OnMouseDown	"MouseDown" event handler
OnOnMouseUp	"MouseUp" event handler

Create

Creates new CButton control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

Pressed (Get method)

Gets the state ("Pressed" property) of the control.

```
bool  Pressed()  const
```

Returned value

Control state.

Pressed (Set method)

Sets the state ("Pressed" property) of the control.

```
bool  Pressed(
   const bool  pressed      // new state
   )
```

Parameters

pressed

[in] New control state.

Returned value

true if successful, otherwise false.

Locking (Get method)

Gets the "Locking" property of the control.

```
bool  Locking()  const
```

Returned value

The value of "Locking" property.

Locking (Set method)

Sets new value of "Locking" property of the control.

```
void  Locking(
   const bool  locking      // new value
   )
```

Parameters

locking

[in] New value of "Locking" property.

Returned value

None.

OnSetText

The control "SetText" (change of the OBJPROP_TEXT property) event handler.

```
virtual bool  OnSetText()
```

Returned value

true if the event has been processed, otherwise false.

OnSetColor

The control "SetColor" (change of the OBJPROP_COLOR property) event handler.

```
virtual bool  OnSetColor()
```

Returned value

true if the event has been processed, otherwise false.

OnSetColorBackground

The control "SetColorBackground" (change of the OBJPROP_BGCOLOR property) event handler.

```
virtual bool  OnSetColorBackground()
```

Returned value

true if the event has been processed, otherwise false.

OnSetColorBorder

The control "SetColorBorder" (change of the OBJPROP_BORDER_COLOR property) event handler.

```
virtual bool  OnSetColorBackground()
```

Returned value

true if the event has been processed, otherwise false.

OnSetFont

The control "SetFont" (change of the OBJPROP_FONT property) event handler.

```
virtual bool  OnSetFont()
```

Returned value

true if the event has been processed, otherwise false.

OnSetFontSize

The control "SetFontSize" (change of the OBJPROP_FONTSIZE property) event handler.

```
virtual bool  OnSetFontSize()
```

Returned value

true if the event has been processed, otherwise false.

OnCreate

The control "Create" event handler.

```
virtual bool  OnCreate()
```

Returned value

true if the event has been processed, otherwise false.

OnShow

The control "Show" event handler.

```
virtual bool  OnShow()
```

Returned value

true if the event has been processed, otherwise false.

OnHide

The control "Hide" event handler.

```
virtual bool  OnHide()
```

Returned value

true if the event has been processed, otherwise false.

OnMove

The control "Move" event handler.

```
virtual bool  OnMove()
```

Returned value

true if the event has been processed, otherwise false.

OnResize

The control "Resize" event handler.

```
virtual bool  OnResize()
```

Returned value

true if the event has been processed, otherwise false.

OnMouseDown

The control "MouseDown" event handler.

```
virtual bool  OnMouseDown()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "MouseDown" event occurs when left mouse button is pressed on the control.

OnMouseUp

The control "MouseUp" (left mouse button release) event handler.

```
virtual bool  OnMouseUp()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "MouseUp" event occurs when left mouse button is released on the control.

CEdit

CEdit is class of the simple control, based on "Edit" chart object.

Description

CEdit class in intended for creation of controls, in which the user can enter text.

Declaration

```
class CEdit : public CWndObj
```

Title

```
#include <Controls\Edit.mqh>
```

Class Methods

Create	
Create	Creates control
Properties	
ReadOnly	Gets/Sets the "ReadOnly" property
TextAlign	Gets/Sets the "TextAlign" property
Chart object event handlers	
OnObjectEndEdit	The CHARTEVENT_OBJECT_ENDEDIT event handler (virtual)
Properties change event handlers	
OnSetText	"SetText" event handler
OnSetColor	"SetColor" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetColorBorder	"SetColorBorder" event handler
OnSetFont	"SetFont" event handler
OnSetFontSize	"SetFontSize" event handler
OnSetZOrder	"SetZOrder" event handler
Internal event handlers	
OnCreate	"Create" event handler
OnShow	"Show" event handler
OnHide	"Hide" event handler
OnMove	"Move" event handler

OnResize	"Resize" event handler
OnChange	"Change" event handler
OnClick	"Click" event handler

Create

Creates new CEdit control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

ReadOnly (Get Method)

Gets the "ReadOnly" property of the control.

```
bool  ReadOnly()
```

Returned value

The value of "ReadOnly" property.

ReadOnly (Set Method)

Sets the value of "ReadOnly" property of the control.

```
bool  ReadOnly(
   const bool  flag      // new values
   )
```

Parameters

flag

[in] New value of "ReadOnly" property.

Returned value

true if successful, otherwise false.

OnObjectEndEdit

The CHARTEVENT_OBJECT_ENDEDIT event handler.

```
virtual bool  OnObjectEndEdit()
```

Returned value

true if the event has been processed, otherwise false.

OnSetText

The control "SetText" (change of the OBJPROP_TEXT property) event handler.

```
virtual bool  OnSetText()
```

Returned value

true if the event has been processed, otherwise false.

OnSetColor

The control "SetColor" (change of the OBJPROP_COLOR property) event handler.

```
virtual bool  OnSetColor()
```

Returned value

true if the event has been processed, otherwise false.

OnSetColorBackground

The control "SetColorBackground" (change of the OBJPROP_BGCOLOR property) event handler.

```
virtual bool  OnSetColorBackground()
```

Returned value

true if the event has been processed, otherwise false.

OnSetColorBorder

The control "SetColorBorder" (change of the OBJPROP_BORDER_COLOR property) event handler.

```
virtual bool  OnSetColorBackground()
```

Returned value

true if the event has been processed, otherwise false.

OnSetFont

The control "SetFont" (change of the OBJPROP_FONT property) event handler.

```
virtual bool  OnSetFont()
```

Returned value

true if the event has been processed, otherwise false.

OnSetFontSize

The control "SetFontSize" (change of the OBJPROP_FONTSIZE property) event handler.

```
virtual bool  OnSetFontSize()
```

Returned value

true if the event has been processed, otherwise false.

OnSetZOrder

The control "SetZOrder" (change of the OBJPROP_ZORDER property) event handler.

```
virtual bool  OnSetZOrder()
```

Returned value

true if the event has been processed, otherwise false.

OnCreate

The control "Create" event handler.

```
virtual bool  OnCreate()
```

Returned value

true if the event has been processed, otherwise false.

OnShow

The control "Show" event handler.

```
virtual bool  OnShow()
```

Returned value

true if the event has been processed, otherwise false.

OnHide

The control "Hide" event handler.

```
virtual bool  OnHide()
```

Returned value

true if the event has been processed, otherwise false.

OnMove

The control "Move" event handler.

```
virtual bool  OnMove()
```

Returned value

true if the event has been processed, otherwise false.

OnResize

The control "Resize" event handler.

```
virtual bool  OnResize()
```

Returned value

true if the event has been processed, otherwise false.

OnChange

The control "Change" event handler.

```
virtual bool  OnChange()
```

Returned value

true if the event has been processed, otherwise false.

OnClick

The control "Click" (left mouse button click) event handler.

```
virtual bool  OnClick()
```

Returned value

true if the event has been processed, otherwise false.

CPanel

CPanel is class of the simple control, based on "Rectangle label" chart object.

Description

CPanel class is intended to combine the controls with similar functions in the group.

Declaration

```
class CPanel : public CWndObj
```

Title

```
#include <Controls\Panel.mqh>
```

Class Methods

Create	
Create	Creates control
Chart object properties	
BorderType	Gets the "BorderType" property of the chart object
Chart object event handlers	
OnSetText	"SetText" event handler
OnSetColorBackground	"SetColorBackground" event handler
OnSetColorBorder	"SetColorBorder" event handler
Internal event handlers	
OnCreate	"Create" event handler
OnShow	"Show" event handler
OnHide	"Hide" event handler
OnMove	"Move" event handler
OnResize	"Resize" event handler
OnChange	"Change" event handler

Create

Creates new CPanel control.

```
virtual bool  Create(
   const long     chart,      // chart ID
   const string   name,       // name
   const int      subwin,     // chart subwindow
   const int      x1,         // x1 coordinate
   const int      y1,         // y1 coordinate
   const int      x2,         // x2 coordinate
   const int      y2          // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

BorderType (Get method)

Gets the "BorderType" property of the chart object.

```
ENUM_BORDER_TYPE  BorderType()
```

Returned value

The value of "BorderType" property.

BorderType (Set method)

Sets new value of "BorderType" property of the chart object.

```
bool  BorderType(
   const ENUM_BORDER_TYPE  type      // value
   )
```

Parameters

type

[in] New value of "BorderType" property.

Returned value

true if successful, otherwise false.

OnSetText

The control "SetText" (change of the OBJPROP_TEXT property) event handler.

```
virtual bool  OnSetText()
```

Returned value

true if the event has been processed, otherwise false.

OnSetColorBackground

The control "SetColorBackground" (change of the OBJPROP_BGCOLOR property) event handler.

```
virtual bool  OnSetColorBackground()
```

Returned value

true if the event has been processed, otherwise false.

OnSetColorBorder

The control "SetColorBorder" (change of the OBJPROP_BORDER_COLOR property) event handler.

```
virtual bool  OnSetColorBackground()
```

Returned value

true if the event has been processed, otherwise false.

OnCreate

The control "Create" event handler.

```
virtual bool  OnCreate()
```

Returned value

true if the event has been processed, otherwise false.

OnShow

The control "Show" event handler.

```
virtual bool  OnShow()
```

Returned value

true if the event has been processed, otherwise false.

OnHide

The control "Hide" event handler.

```
virtual bool  OnHide()
```

Returned value

true if the event has been processed, otherwise false.

OnMove

The control "Move" event handler.

```
virtual bool  OnMove()
```

Returned value

true if the event has been processed, otherwise false.

OnResize

The control "Resize" event handler.

```
virtual bool  OnResize()
```

Returned value

true if the event has been processed, otherwise false.

OnChange

The control "Change" event handler.

```
virtual bool  OnChange()
```

Returned value

true if the event has been processed, otherwise false.

CPicture

CPicture is class of the simple control, based on "Bitmap Label" chart object.

Description

CPicture class is intended for creation of simple graphic images.

Declaration

```
class CPicture : public CWndObj
```

Title

```
#include <Controls\Picture.mqh>
```

Class Methods

Create	
Create	Creates control
Chart object properties	
Border	Gets/Sets the border width of the chart object
BmpName	Gets/Sets the name of bmp-file of the control
Internal events	
OnCreate	"Create" event handler
OnShow	"Show" event handler
OnHide	"Hide" event handler
OnMove	"Move" event handler
OnChange	"Change" event handler

Create

Creates new CPicture control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

Border (Get method)

Gets the "Border" (border width) property of the control.

```
int  Border()  const
```

Returned value

The "Border" property.

Border (Set method)

Sets the "Border" (border width) property of the control.

```
bool  Border(
   const int  value      // new value
   )
```

Parameters

value

[in] New value of the "Border" property.

Returned value

true if successful, otherwise false.

BmpName (Get method)

Gets the name of bmp file of the control.

```
string  BmpName()  const
```

Returned value

Name of bmp file of the control.

BmpName (Set method)

Sets the name of bmp file of the control.

```
bool  BmpName(
   const string  name      // file name
   )
```

Parameters

name

[in] Name of bmp file of the control.

Returned value

true if successful, otherwise false.

OnCreate

The control "Create" event handler.

```
virtual bool  OnCreate()
```

Returned value

true if the event has been processed, otherwise false.

OnShow

The control "Show" event handler.

```
virtual bool  OnShow()
```

Returned value

true if the event has been processed, otherwise false.

OnHide

The control "Hide" event handler.

```
virtual bool  OnHide()
```

Returned value

true if the event has been processed, otherwise false.

OnMove

The control "Move" event handler.

```
virtual bool  OnMove()
```

Returned value

true if the event has been processed, otherwise false.

OnChange

The control "Change" event handler.

```
virtual bool  OnChange()
```

Returned value

true if the event has been processed, otherwise false.

CScroll

CScroll is a base class for creation of scroll bars.

Description

CScroll is a complex control (with dependent controls), it contains the base functionality for creation of scroll bars. The base class itself isn't used as a separate control, two of its heirs (CScrollV and CScrollH classes) are used as controls.

Declaration

```
class CScroll : public CWndContainer
```

Title

```
#include <Controls\Scrolls.mqh>
```

Class Methods

Create	
Create	Creates control
Chart object event handlers	
OnEvent	Event handler of all chart events
Properties	
MinPos	Gets/Sets the minimal position
MaxPos	Gets/Sets the maximal position
CurrPos	Gets/Sets the current position
Dependent controls creation	
CreateBack	Creates background button
CreateInc	Creates increment button of the scroll bar
CreateDec	Creates decrement button of the scroll bar
CreateThumb	Creates thumb button (can be dragged) of the scroll bar
Dependent controls event handlers	
OnClickInc	Event handler, used for handling of increment button events
OnClickDec	Event handler, used for handling of decrement button events
Internal event handlers	
OnShow	"Create" event handler

OnHide	"Hide" event handler
OnChangePos	"ChangePosition" event handler
Object drag handlers	
OnThumbDragStart	"ThumbDragStart" event handler
OnThumbDragProcess	"ThumbDragProcess" event handler
OnThumbDragEnd	"ThumbDragEnd" event handler
Position	
CalcPos	Gets scroll bar postion by coordinate

Create

Creates new CScroll control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

MinPos (Get method)

Gets the value of "MinPos" (minimal position) of the CScroll control.

```
int  MinPos()  const
```

Returned value

New value of "MinPos" property.

MinPos (Set method)

Sets the value of "MinPos" (minimal position) of the CScroll control.

```
void  MinPos(
   const int  value      // new value
   )
```

Parameters

value

[in] New value of "MinPos" property.

Returned value

None.

MaxPos (Get method)

Gets the value of "MinPos" (maximal position) of the CScroll control.

```
int  MaxPos()  const
```

Returned value

New value of "MaxPos" property.

MaxPos (Set method)

Sets the value of "MaxPos" (maximal position) of the CScroll control.

```
void  MaxPos(
   const int  value      // new value
   )
```

Parameters

value

[in] New value of "MaxPos" property.

Returned value

None.

CurrPos (Get method)

Gets the value of "CurrPos" (current position) of the CScroll control.

```
int  CurrPos()  const
```

Returned value

New value of "MaxPos" property.

CurrPos (Set method)

Sets the value of "CurrPos" (current position) of the CScroll control.

```
void  CurrPos(
   const int  value      // new value
   )
```

Parameters

value

[in] New value of "CurrPos" property.

Returned value

None.

CreateBack

Creates background button of the CScroll control.

```
virtual bool  CreateBack()
```

Returned value

true if successful, otherwise false.

CreateInc

Creates increment button of the CScroll control.

```
virtual bool  CreateInc()
```

Returned value

true if successful, otherwise false.

CreateDec

Creates decrement button of CScroll control.

```
virtual bool  CreateDec()
```

Returned value

true if successful, otherwise false.

CreateThumb

Creates thumb button (can be dragged) of the CScroll control.

```
virtual bool  CreateThumb()
```

Returned value

true if successful, otherwise false.

OnClickInc

The control "ClickInc" (left mouse button click on the increment button) event handler.

```
virtual bool  OnClickInc()
```

Returned value

true if the event has been processed, otherwise false.

OnClickDec

The control "ClickDec" (left mouse button click on the decrement button) event handler.

```
virtual bool  OnClickDec()
```

Returned value

true if the event has been processed, otherwise false.

OnShow

The control "Show" event handler.

```
virtual bool  OnShow()
```

Returned value

true if the event has been processed, otherwise false.

OnHide

The control "Hide" event handler.

```
virtual bool  OnHide()
```

Returned value

true if the event has been processed, otherwise false.

OnChangePos

The control "ChangePos" (position change) event handler.

```
virtual bool  OnChangePos()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnThumbDragStart

The control "ThumbDragStart" (drag start) event handler.

```
virtual bool  OnThumbDragStart()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "ThumbDragStart" event occurs at start of the drag operation.

OnThumbDragProcess

The control "ThumbDragProcess" event handler.

```
virtual bool  OnThumbDragProcess(
   const int  x,     // x coordinate
   const int  y      // y coordinate
   )
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Returned value

true if the event has been processed, otherwise false.

Note

The "ThumbDragProcess" occurs when the scroll bar control (thumb button) is moved.

OnThumbDragEnd

The control "ThumbDragEnd" (drag process finished) event handler.

```
virtual bool  OnThumbDragEnd()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "ThumbDragEnd" occurs when the drag operation of the scroll bar control (thumb button) is finished.

CalcPos

Gets scroll bar position by coordinate.

```
virtual int  CalcPos(
   const int      coord      // coordinate
   )
```

Parameters

coord

[in] Scroll bar coordinate.

Returned value

Scroll bar position.

CScrollV

CScrollV is a class of the "Vertical scroll bar" complex control.

Description

CScrollV class is intended for creation of vertical scroll bars.

Declaration

```
class CScrollV : public CScroll
```

Title

```
#include <Controls\Scrolls.mqh>
```

Class Methods

Dependent controls	
CreateInc	Creates scroll bar increment button
CreateDec	Creates scroll bar decrement button
CreateThumb	Creates scroll bar thumb button (can be dragged)
Internal event handlers	
OnResize	"Resize" event handler
OnChangePos	"ChangePosition" event handler
Drag event handlers	
OnThumbDragStart	"ThumbDragStart" event handler
OnThumbDragProcess	"ThumbDragProcess" event handler
OnThumbDragEnd	"ThumbDragEnd" event handler
Position	
CalcPos	Gets scroll bar postion by coordinate

CreateInc

Creates increment button of the control.

```
virtual bool  CreateInc()
```

Returned value

true if successful, otherwise false.

CreateDec

Creates decrement button of the control.

```
virtual bool  CreateDec()
```

Returned value

true if successful, otherwise false.

CreateThumb

Creates thumb button (can be dragged) of the control.

```
virtual bool  CreateThumb()
```

Returned value

true if successful, otherwise false.

OnResize

The control "Resize" event handler.

```
virtual bool  OnResize()
```

Returned value

true if the event has been processed, otherwise false.

OnChangePos

The control "ChangePos" (position change) event handler.

```
virtual bool  OnChangePos()
```

Returned value

true if the event has been processed, otherwise false.

OnThumbDragStart

The control "ThumbDragStart" (drag start) event handler.

```
virtual bool  OnThumbDragStart()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "ThumbDragStart" event occurs at start of the drag operation.

OnThumbDragProcess

The control "ThumbDragProcess" event handler.

```
virtual bool  OnThumbDragProcess(
   const int  x,     // x coordinate
   const int  y      // y coordinate
   )
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Returned value

true if the event has been processed, otherwise false.

Note

The "ThumbDragProcess" occurs when the scroll bar control (thumb button) is moved.

OnThumbDragEnd

The control "ThumbDragEnd" (drag process finished) event handler.

```
virtual bool  OnThumbDragEnd()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "ThumbDragEnd" occurs when the drag operation of the scroll bar control (thumb button) is finished.

CalcPos

Gets scroll bar position by coordinate.

```
virtual int  CalcPos(
   const int      coord      // coordinate
   )
```

Parameters

coord

[in] Scroll bar coordinate.

Returned value

Scroll bar position.

CScrollH

CScrollH is a class of the "Horizontal scroll bar" complex control.

Description

CScrollH is intended for creation of horizontal scroll bars.

Declaration

```
class CScrollH : public CScroll
```

Title

```
#include <Controls\Scrolls.mqh>
```

Class Methods

Dependent controls	
CreateInc	Creates scroll bar increment button
CreateDec	Creates scroll bar decrement button
CreateThumb	Creates scroll bar thumb button (can be dragged)
Internal event handlers	
OnResize	"Resize" event handler
OnChangePos	"ChangePosition" event handler
Drag event handlers	
OnThumbDragStart	"ThumbDragStart" event handler
OnThumbDragProcess	"ThumbDragProcess" event handler
OnThumbDragEnd	"ThumbDragEnd" event handler
Position	
CalcPos	Gets scroll bar postion by coordinate

CreateInc

Creates increment button of the control.

```
virtual bool  CreateInc()
```

Returned value

true if successful, otherwise false.

CreateDec

Creates decrement button of the control.

```
virtual bool  CreateDec()
```

Returned value

true if successful, otherwise false.

CreateThumb

Creates thumb button (can be dragged) of the control.

```
virtual bool  CreateThumb()
```

Returned value

true if successful, otherwise false.

OnResize

The control "Resize" event handler.

```
virtual bool  OnResize()
```

Returned value

true if the event has been processed, otherwise false.

OnChangePos

The control "ChangePos" (position change) event handler.

```
virtual bool  OnChangePos()
```

Returned value

true if the event has been processed, otherwise false.

OnThumbDragStart

The control "ThumbDragStart" (drag start) event handler.

```
virtual bool  OnThumbDragStart()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "ThumbDragStart" event occurs at start of the drag operation.

OnThumbDragProcess

The control "ThumbDragProcess" event handler.

```
virtual bool  OnThumbDragProcess(
   const int  x,     // x coordinate
   const int  y      // y coordinate
   )
```

Parameters

x

[in] Current X coordinate of mouse cursor.

y

[in] Current Y coordinate of mouse cursor.

Returned value

true if the event has been processed, otherwise false.

Note

The "ThumbDragProcess" occurs when the scroll bar control (thumb button) is moved.

OnThumbDragEnd

The control "ThumbDragEnd" (drag process finished) event handler.

```
virtual bool  OnThumbDragEnd()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "ThumbDragEnd" occurs when the drag operation of the scroll bar control (thumb button) is finished.

CalcPos

Gets scroll bar position by coordinate.

```
virtual int  CalcPos(
   const int      coord      // coordinate
   )
```

Parameters

coord

[in] Scroll bar coordinate.

Returned value

Scroll bar position.

CWndClient

CWndClient is a class of the "Cient area" complex control (with dependent controls). It's a base class for creation of scroll bars area.

Description

CWndClient implements the functionality for creation of client area with scroll bars.

Declaration

```
class CWndClient : public CWndContainer
```

Title

```
#include <Controls\WndClient.mqh>
```

Class Methods

Create	
Create	Creates control
Chart event handler	
OnEvent	Event handler of all chart events
Properties	
ColorBackground	Sets background color
ColorBorder	Sets border color
BorderType	Sets border type
Settings	
VScrolled	Gets/Sets the flag, indicating that vertical scroll bar is used
HScrolled	Gets/Sets the flag, indicating that horizontal scroll bar is used
Dependent controls	
CreateBack	Creates background for scroll bar
CreateScrollV	Creates vertical scroll bar
CreateScrollH	Creates horizontal scroll bar
Internal event handlers	
OnResize	"Resize" event handler
Dependent controls event handlers	
OnVScrollShow	"Show" event handler (virtual) of VScroll

	dependent control
OnVScrollHide	"Hide" event handler (virtual) of VScroll dependent control
OnHScrollShow	"Show" event handler (virtual) of HScroll dependent control
OnHScrollHide	"Hide" event handler (virtual) of HScroll dependent control
OnScrollLineDown	"ScrollLineDown" event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineUp" event handler (virtual) of VScroll dependent control
OnScrollLineLeft	"ScrollLineLeft" event handler (virtual) of HScroll dependent control
OnScrollLineRight	"ScrollLineRight" event handler (virtual) of HScroll dependent control
Resize	
Rebound	Sets new coordinates of the control using CRect class coordinates

Create

Creates new CWndClient control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

ColorBackground

Sets background color of the control.

```
bool  ColorBackground(
   const color  value      // new color
   )
```

Parameters

value

[in] New background color of the control.

Returned value

true if successful, otherwise false.

ColorBorder

Set border color of the control.

```
bool  ColorBorder(
   const color  value      // color
   )
```

Parameters

value

[in] New border color of the control.

Returned value

true if successful, otherwise false.

BorderType

Sets border type of the control.

```
bool  BorderType(
   const ENUM_BORDER_TYPE  type      // border type
   )
```

Parameters

type

[in] Border type of the control.

Returned value

true if successful, otherwise false.

VScrolled (Get method)

Gets the flag, indicating that vertical scroll bar is used.

```
bool  VScrolled()
```

Returned value

true, if vertical scroll bar is used, otherwise false.

VScrolled (Set method)

Sets the flag, indicating that vertical scroll bar is used.

```
bool  VScrolled(
   const bool  flag      // flag
   )
```

Parameters

flag

[in] Flag.

Returned value

true if successful, otherwise false.

HScrolled (Get method)

Gets the flag, indicating that horizontal scroll bar is used.

```
bool  HScrolled()
```

Returned value

true, if horizontal scroll bar is used, otherwise false.

HScrolled (Set method)

Sets the flag, indicating that horizontal scroll bar is used.

```
bool  HScrolled(
   const bool  flag      // flag
   )
```

Parameters

flag

[in] Flag.

Returned value

true if successful, otherwise false.

CreateBack

Creates background button of the control.

```
virtual bool  CreateBack()
```

Returned value

true if successful, otherwise false.

CreateScrollV

Creates vertical scroll bar.

```
virtual bool  CreateScrollV()
```

Returned value

true if successful, otherwise false.

CreateScrollH

Creates horizontal scroll bar.

```
virtual bool  CreateScrollH()
```

Returned value

true if successful, otherwise false.

OnResize

The control "Resize" event handler.

```
virtual bool  OnResize()
```

Returned value

true if the event has been processed, otherwise false.

OnVScrollShow

The control "VScrollShow" (vertical scroll bar show) event handler.

```
virtual bool  OnVScrollShow()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnVScrollHide

The control "VScrollHide" (vertical scroll bar hide) event handler.

```
virtual bool  OnVScrollHide()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnHScrollShow

The control "HScrollShow" (horizontal scroll bar show) event handler.

```
virtual bool  OnHScrollShow()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnHScrollHide

The control "HScrollHide" (vertical scroll bar hide) event handler.

```
virtual bool  OnHScrollHide()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnScrollLineDown

The control "ScrollLineDown" (vertical scroll line down) event handler.

```
virtual bool  OnScrollLineDown()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnScrollLineUp

The control "ScrollLineUp" (vertical scroll line up) event handler.

```
virtual bool  OnScrollLineUp()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnScrollLineLeft

The control "ScrollLineLeft" (horizontal scroll line left) event handler.

```
virtual bool  OnScrollLineLeft()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

OnScrollLineRight

The control "ScrollLineRight" (horizontal scroll line right) event handler.

```
virtual bool  OnScrollLineRight()
```

Returned value

true if the event has been processed, otherwise false.

Note

The base class method does nothing and always returns true.

ReBound

Sets new coordinates of the control using CRect class coordinates.

```
void  ReBound(
   const & CRect  rect      // CRect class
   )
```

Returned value

None.

CListView

CListView is a class of the ListView complex control (with dependent controls).

Description

CListView class encapsulates list-control functionality.

Declaration

```
class CListView : public CWndClient
```

Title

```
#include <Controls\ListView.mqh>
```

Class Methods

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Settings	
TotalView	Sets the number of items, shown on the control
Add/Delete	
AddItem	Adds an item
Data	
Select	Selects current list item by index
SelectByText	Selects current list item by text
SelectByValue	Selects current list item by value
Read-only data	
Value	Gets the value of current list item
Dependent controls	
CreateRow	Creates a row of the ListView
Internal event handlers	
OnResize	"Resize" event handler (virtual)
Dependent controls event handlers	
OnVScrollShow	"Show" event handler (virtual) of VScroll dependent control

OnVScrollHide	"Hide" event handler (virtual) of VScroll dependent control
OnScrollLineDown	"ScrollLineDown" event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineUp" event handler (virtual) of VScroll dependent control
OnItemClick	"ItemClick" event handler (virtual)
Redraw	
Redraw	Redraws the control
RowState	Sets the state of the specified row
CheckView	Checks the "visibility" of the specified row

Create

Creates new CListView control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

TotalView

Sets the number of items, shown on the control.

```
bool  TotalView(
   const int  value      // items shown
   )
```

Parameters

value

[in] The number of items, shown on the control.

Returned value

true if successful, otherwise false.

Note

The number of shown items can be specified only at once.

AddItem

Adds an item to the control.

```
bool  AddItem(
   const string  item,     // text
   const long    value     // value
   )
```

Parameters

item

[in] Text.

value

[in] Value of long type.

Returned value

true if successful, otherwise false.

Select

Selects current list item by index.

```
bool  Select(
   const int  index     // index
   )
```

Parameters

index

[in] Item index.

Returned value

true if successful, otherwise false.

SelectByText

Selects current list item by text.

```
bool  SelectByText(
   const string  text     // text
   )
```

Parameters

text

[in] Text.

Returned value

true if successful, otherwise false.

SelectByValue

Selects current list item by value.

```
bool  SelectByValue(
   const long  value     // value
   )
```

Parameters

value

[in] Value of long type.

Returned value

true if successful, otherwise false.

Value

Gets the value of current list item.

```
long  Value()
```

Returned value

The value of current list item.

CreateRow

Creates a row of the CListView control.

```
bool  CreateRow(
   const int  index     // index
   )
```

Parameters

index

[in] Item index.

Returned value

true if successful, otherwise false.

OnResize

The control "Resize" event handler.

```
virtual bool  OnResize()
```

Returned value

true if the event has been processed, otherwise false.

OnVScrollShow

The control "VScrollShow" (vertical scroll bar show) event handler.

```
virtual bool  OnVScrollShow()
```

Returned value

true if the event has been processed, otherwise false.

OnVScrollHide

The control "VScrollHide" (vertical scroll bar hide) event handler.

```
virtual bool  OnVScrollHide()
```

Returned value

true if the event has been processed, otherwise false.

OnScrollLineDown

The control "ScrollLineDown" (vertical scroll line down) event handler.

```
virtual bool  OnScrollLineDown()
```

Returned value

true if the event has been processed, otherwise false.

OnScrollLineUp

The control "ScrollLineUp" (vertical scroll line up) event handler.

```
virtual bool  OnScrollLineUp()
```

Returned value

true if the event has been processed, otherwise false.

OnItemClick

The control "ItemClick" (mouse button click on a row) event handler.

```
virtual bool  OnItemClick()
   const int    index     // row index
   )
```

Returned value

true if the event has been processed, otherwise false.

Redraw

Redraws the control.

```
bool  Redraw()
```

Returned value

true if successful, otherwise false.

RowState

Sets the state of the specified row.

```
bool  RowState(
   const int   index      // index
   const bool  select     // state
   )
```

Parameters

index

[in] Row index.

select

[in] Row state.

Returned value

true if successful, otherwise false.

CheckView

Checks the "visibility" of the specified row.

```
bool  CheckView()
```

Returned value

true, if the selected row is visible, othewise false.

CComboBox

CComboBox is a class of the ComboBox complex control (with dependent controls).

Description

ComboBox consists of a list box, combined with a static control, intended for selection. The list-box portion of the control may be drop down when the user selects the drop-down arrow next to the control

Declaration

```
class CComboBox : public CWndContainer
```

Title

```
#include <Controls\ComboBox.mqh>
```

Class Methods

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Add	
AddItem	Adds an item
Settings	
ListViewItems	Sets the number of items, shown on the control
Data	
Select	Selects current list item by index
SelectByText	Selects current list item by text
SelectByValue	Selects current list item by value
Read-only data	
Value	Gets the value of the current list item
Dependent controls	
CreateEdit	Creates dependent control (edit)
CreateButton	Creates dependent control (button)
CreateList	Creates dependent control (list view)
Dependent controls event handlers	
OnClickEdit	"ClickEdit" event handler (virtual)

OnClickButton	"ClickButton" event handler (virtual)
OnChangeList	"ChangeList" event handler (virtual)
Show/Hide	
ListShow	Shows the items list
ListHide	Hides the items list

Create

Creates new CComboBox control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

AddItem

Adds an item to the control.

```
bool  AddItem(
   const string  item,      // text
   const long    value      // value
   )
```

Parameters

item

[in] Text.

value=0

[in] Value of long type.

Returned value

true if successful, otherwise false.

ListViewItems

Sets the number of list items of the CComboBox control.

```
void  ListViewItems(
   const int     value      // number of list items
   )
```

Parameters

value

[in] Number of list items.

Returned value

true if successful, otherwise false.

Select

Selects current list item by index.

```
bool  Select(
   const int  index     // index
   )
```

Parameters

index

[in] Item index.

Returned value

true if successful, otherwise false.

SelectByText

Selects current list item by text.

```
bool  SelectByText(
   const string  text     // text
   )
```

Parameters

text

[in] Text of the item.

Returned value

true if successful, otherwise false.

SelectByValue

Selects current list item by value.

```
bool  SelectByValue(
   const long  value     // value
   )
```

Parameters

value

[in] Value of long type.

Returned value

true if successful, otherwise false.

Value

Gets the value of the current list item.

```
long  Value()
```

Returned value

The value of the current list item.

CreateEdit

Creates dependent control (edit) of the control.

```
virtual bool  CreateEdit()
```

Returned value

true if successful, otherwise false.

CreateButton

Creates dependent control (button).

```
virtual bool  CreateButton()
```

Returned value

true if successful, otherwise false.

CreateList

Creates dependent control (list view).

```
virtual bool  CreateList()
```

Returned value

true if successful, otherwise false.

OnClickEdit

The control "ClickEdit" (mouse click on the edit) event handler.

```
virtual bool  OnClickEdit()
```

Returned value

true if the event has been processed, otherwise false.

OnClickButton

The control "ClickButton" (mouse click on the button) event handler.

```
virtual bool  OnClickButton()
```

Returned value

true if the event has been processed, otherwise false.

OnChangeList

The "ChangeList" (change of the list) event handler.

```
virtual bool  OnChangeList()
```

Returned value

true if the event has been processed, otherwise false.

ListShow

Shows the items list.

```
virtual bool  ListShow()
```

Returned value

true if successful, otherwise false.

ListHide

Hides the items list.

```
virtual bool  ListHide()
```

Returned value

true if successful, otherwise false.

CCheckBox

CCheckBox is class of the CheckBox complex control.

Description

CCheckBox control displays a check box that allows the user to select a true or false condition.

Declaration

```
class CCheckBox : public CWndContainer
```

Title

```
#include <Controls\CheckBox.mqh>
```

Class Methods

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Properties	
Text	Gets/Sets text label, associated with the control
Color	Gets/Sets color of text label, associated with the control
State	
Checked	Gets/Sets a value indicating whether the control is checked
Data	
Value	Gets/Sets the value associated with the control
Dependent controls	
CreateButton	Creates dependent control (button)
CreateLabel	Creates dependent control (label)
Dependent controls event handlers	
ClickButton	"ClickButton" event handler (virtual)
ClickLabel	"ClickLabel" event handler (virtual)

Create

Creates new CCheckBox control.

```
virtual bool  Create(
   const long     chart,      // chart ID
   const string   name,       // name
   const int      subwin,     // chart subwindow
   const int      x1,         // x1 coordinate
   const int      y1,         // y1 coordinate
   const int      x2,         // x2 coordinate
   const int      y2          // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

Text (Get method)

Gets text of the label, associated with the control.

```
string  Text()
```

Returned value

Text of the label.

Text (Set method)

Sets text of the label, associated with the control.

```
bool  Text(
   const string  value      // text
   )
```

Parameters

value

[in] New text of the label.

Returned value

true if successful, otherwise false.

Color (Get method)

Gets color of the label, associated with the control.

```
color  Color()  const
```

Returned value

Label color.

Color (Set method)

Sets color of the label, associated with the control.

```
bool  Color(
   const color  value      // color
   )
```

Parameters

value

[in] New label color.

Returned value

true if successful, otherwise false.

Checked (Get method)

Gets state of the control.

```
bool  Checked()  const
```

Returned value

State of the control.

Checked (Set method)

Sets state of the control.

```
bool  Checked(
   const bool  flag      // state
   )
```

Parameters

flag

[in] New state.

Returned value

true if successful, otherwise false.

Value (Get method)

Gets the value, associated with the control.

```
int  Value()  const
```

Returned value

The value, associated with the control.

Value (Set method)

Sets the value, associated with the control.

```
void  Value(
   const int  value      // new value
   )
```

Parameters

value

[in] New value.

Returned value

None.

CreateButton

Creates dependent control (button).

```
virtual bool  CreateButton()
```

Returned value

true if successful, otherwise false.

CreateLabel

Creates dependent control (label).

```
virtual bool  CreateLabel()
```

Returned value

true if successful, otherwise false.

OnClickButton

The control "ClickButton" (mouse click on the button) event handler.

```
virtual bool  OnClickButton()
```

Returned value

true if the event has been processed, otherwise false.

OnClickLabel

The control "ClickLabel" (mouse click on the label) event handler.

```
virtual bool  OnClickLabel()
```

Returned value

true if the event has been processed, otherwise false.

CCheckGroup

CCheckGroup is a class of the CheckGroup complex control (with dependent controls).

Description

CCheckGroup provides the possibility for creation of controls, which allow to display and edit flags.

Declaration

```
class CCheckGroup : public CWndClient
```

Title

```
#include <Controls\CheckGroup.mqh>
```

Class Methods

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler for all chart events
Add	
AddItem	Adds new item
Read-only data	
Value	Gets the value, associated with the control
Dependent controls	
CreateButton	Creates new CCheckBox item
Dependent controls event handlers	
OnVScrollShow	"Show" event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" event handler (virtual) of VScroll dependent control
OnScrollLineDown	"ScrollLineUp" event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineDown" event handler (virtual) of VScroll dependent control
OnChangeItem	"ChangeItem" event handler (virtual) of VScroll dependent control
Redraw	
Redraw	Redraws the group

RowState	Sets the state of the specified item

Create

Creates new CCheckGroup control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

AddItem

Adds an item to the control.

```
bool  AddItem(
   const string  item,      // text
   const long    value      // value
   )
```

Parameters

item

[in] Text.

value=0

[in] Value of long type.

Returned value

true if successful, otherwise false.

Value

Gets the value, associated with the control.

```
long  Value()
```

Returned value

The value, associated with the control.

Note

The value depends on state of all items of the CCheckGroup.

CreateButton

Creates new CCheckBox class instance on the specified index.

```
bool  CreateButton(
   int  index      // index
   )
```

Parameters

index

[in] Index of the new item in the CCheckGroup.

Returned value

true if successful, otherwise false.

OnVScrollShow

The control "VScrollShow" (vertical scroll bar show) event handler.

```
virtual bool  OnVScrollShow()
```

Returned value

true if the event has been processed, otherwise false.

OnVScrollHide

The control "VScrollHide" (vertical scroll bar hide) event handler.

```
virtual bool  OnVScrollHide()
```

Returned value

true if the event has been processed, otherwise false.

OnScrollLineDown

The control "ScrollLineDown" (vertical scroll line down) event handler.

```
virtual bool  OnScrollLineDown()
```

Returned value

true if the event has been processed, otherwise false.

OnScrollLineUp

The control "ScrollLineUp" (vertical scroll line up) event handler.

```
virtual bool  OnScrollLineUp()
```

Returned value

true if the event has been processed, otherwise false.

OnChangeItem

The control "ChangeItem" (item change) event handler.

```
virtual bool  OnChangeItem(
   const int  index      // index
   )
```

Parameters

index

[in] Index of the changed item.

Returned value

true if the event has been processed, otherwise false.

Redraw

Redraws the control.

```
bool  Redraw()
```

Returned value

true if successful, otherwise false.

RowState

Sets the state of the specified item.

```
bool  RowState(
   const int   index,      // item index
   const bool  select      // state
   )
```

Parameters

index

[in] Item index to change.

select

[in] New state.

Returned value

true if successful, otherwise false.

CRadioButton

CRadioButton is a class of RadioButton complex control.

Description

CRadioButton itself is not used, it used for creation of CRadioGroup items.

Declaration

```
class CRadioButton : public CWndContainer
```

Title

```
#include <Controls\RadioButton.mqh>
```

Class Methods

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler for all chart events
Properties	
Text	Gets/Sets the text label, associated with the control
Color	Gets/Sets the color of text label, associated with the control
State	
State	Gets/Sets the state
Dependent controls	
CreateButton	Creates button
CreateLabel	Creates label
Dependent controls event handlers	
OnClickButton	"ClickButton" event handler (virtual)
OnClickLabel	"ClickLabel" event handler (virtual)

Create

Creates new CRadioButton control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

Text (Get method)

Gets text of the label, associated with the control.

```
string  Text()
```

Returned value

Text of the label.

Text (Set method)

Sets text of the label, associated with the control.

```
bool  Text(
   const string  value      // text
   )
```

Parameters

value

[in] New text of the label.

Returned value

true if successful, otherwise false.

Color (Get method)

Gets color of the label, associated with the control.

```
color  Color()  const
```

Returned value

Label color.

Color (Set method)

Sets color of the label, associated with the control.

```
bool  Color(
   const color  value      // color
   )
```

Parameters

value

[in] New label color.

Returned value

true if successful, otherwise false.

State (Get method)

Gets the button state.

```
bool  State()  const
```

Returned value

Button state.

State (Set method)

Sets the button state.

```
bool  State(
   const bool  flag      // flag
   )
```

Parameters

flag

[in] New button state.

Returned value

true if successful, otherwise false.

CreateButton

Creates button.

```
virtual bool  CreateButton()
```

Returned value

true if successful, otherwise false.

CreateLabel

Creates label.

```
virtual bool  CreateLabel()
```

Returned value

true if successful, otherwise false.

OnClickButton

The control "ClickButton" (mouse click) event handler.

```
virtual bool  OnClickButton()
```

Returned value

true if the event has been processed, otherwise false.

OnClickLabel

The control "ClickLabel" (mouse click on the label) event handler.

```
virtual bool  OnClickLabel()
```

Returned value

true if the event has been processed, otherwise false.

CRadioGroup

CRadioGroup is class of the RadioGroup complex control (with dependent controls).

Description

CRadioGroup enables the user to select a single option from a group of choices when paired with other CRadioButton controls.

Declaration

```
class CRadioGroup : public CWndClient
```

Title

```
#include <Controls\RadioGroup.mqh>
```

Class Methods

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Add	
AddItem	Adds new item
Read-only data	
Value	Gets the value, associated with the control
Dependent controls	
CreateButton	Creates new CRadioButton item
Dependent controls event handlers	
OnVScrollShow	"Show" event handler (virtual) of VScroll dependent control
OnVScrollHide	"Hide" event handler (virtual) of VScroll dependent control
OnScrollLineDown	"ScrollLineDown" event handler (virtual) of VScroll dependent control
OnScrollLineUp	"ScrollLineUp" event handler (virtual) of VScroll dependent control
OnChangeItem	"ChangeItem" event handler (virtual)
Redraw	
Redraw	Redraws the group items

RowState	Sets the state of the specified item
Select	Selects current item

Create

Creates new CRadioGroup control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

AddItem

Adds an item to the control.

```
bool  AddItem(
   const string  item,      // text
   const long    value=0    // value
   )
```

Parameters

item

[in] Text.

value=0

[in] Value of long type.

Returned value

true if successful, otherwise false.

Value

Gets the value, associated with the control.

```
long  Value()
```

Returned value

The value, associated with the control.

Note

The value depends on state of all CRadioButton items of the CRadioGroup control.

CreateButton

Creates new CRadioButton class instance on the specified index.

```
bool  CreateButton(
   const int  index      // index
   )
```

Parameters

index

[in] Index of the new item in the CRadioGroup.

Returned value

true if successful, otherwise false.

OnVScrollShow

The control "VScrollShow" (vertical scroll bar show) event handler.

```
virtual bool  OnVScrollShow()
```

Returned value

true if the event has been processed, otherwise false.

OnVScrollHide

The control "VScrollHide" (vertical scroll bar hide) event handler.

```
virtual bool  OnVScrollHide()
```

Returned value

true if the event has been processed, otherwise false.

OnScrollLineDown

The control "ScrollLineDown" (vertical scroll line down) event handler.

```
virtual bool  OnScrollLineDown()
```

Returned value

true if the event has been processed, otherwise false.

OnScrollLineUp

The control "ScrollLineUp" (vertical scroll line up) event handler.

```
virtual bool  OnScrollLineUp()
```

Returned value

true if the event has been processed, otherwise false.

OnChangeItem

The control "ChangeItem" (item change) event handler.

```
virtual bool  OnChangeItem(
   const int  index      // index
   )
```

Parameters

index

[in] Index of the changed item.

Returned value

true if the event has been processed, otherwise false.

Redraw

Redraws the control.

```
bool  Redraw()
```

Returned value

true if successful, otherwise false.

RowState

Sets the state of the specified item.

```
bool  RowState(
   const int   index,      // item index
   const bool  select      // state
   )
```

Parameters

index

[in] Item index to change.

select

[in] New state.

Returned value

true if successful, otherwise false.

Select

Selects current item.

```
void  Select(
   const int  index      // index
   )
```

Parameters

index

[in] Item index to select.

Returned value

None.

CSpinEdit

CSpinEdit is a class of SpinEdit complex control (with dependent controls).

Description

CSpinEdit class is intended for creation of controls, which allows to edit the values of integer type. It automatically increases data when pressing the upper button, but decreases if pressing the lower button.

Declaration

```
class CSpinEdit : public CWndContainer
```

Title

```
#include <Controls\SpinEdit.mqh>
```

Class Methods

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Properties	
MinValue	Gets/Sets the minimal allowed value
MaxValue	Gets/Sets the maximal allowed value
State	
Value	Gets/Sets the current value
Dependent controls	
CreateEdit	Creates dependent control (edit)
CreateInc	Creates dependent control (increment button)
CreateDec	Creates dependent control (decrement button)
Dependent controls event handlers	
OnClickInc	"ClickInc" event handler (virtual)
OnClickDec	"ClickDec" event handler (virtual)
Internal event handlers	
OnChangeValue	"ChangeValue" event handler (virtual)

Create

Creates new CSpinEdit control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

MinValue (Get method)

Gets the value of "MinValue" property (minimal allowed value) of the control.

```
int  MinValue()  const
```

Returned value

The value of "MinValue" property.

MinValue (Set method)

Sets the value of "MinValue" property (minimal allowed value) of the control.

```
void  MinValue(
   const int  value      // new value
   )
```

Parameters

value

[in] New value of "MinValue" property.

Returned value

None.

MaxValue (Get method)

Gets the value of "MaxValue" property (maximal allowed value) of the control.

```
int  MaxValue()  const
```

Returned value

The value of "MaxValue" property.

MaxValue (Set method)

Sets the value of "MaxValue" property (maximal allowed value) of the control.

```
void  MaxValue(
   const int  value      // new value
   )
```

Parameters

value

[in] New value of "MaxValue" property.

Returned value

None.

Value (Get method)

Gets the "Value" property (current value) of the control.

```
int  Value()  const
```

Returned value

The "Value" property.

Value (Set method)

Sets the "Value" property (current value) of the control.

```
void  Value(
   const int  value      // value
   )
```

Parameters

value

[in] New "Value" property.

Returned value

None.

CreateEdit

Creates dependent control (CEdit).

```
virtual bool  CreateEdit()
```

Returned value

true if successful, otherwise false.

CreateInc

Creates dependent control (increment button).

```
virtual bool  CreateInc()
```

Returned value

true if successful, otherwise false.

CreateDec

Creates dependent control (decrement button).

```
virtual bool  CreateDec()
```

Returned value

true if successful, otherwise false.

OnClickInc

The control "ClickInc" (mouse click on the increment button) event handler.

```
virtual bool  OnClickInc()
```

Returned value

true if the event has been processed, otherwise false.

OnClickDec

The control "ClickDec" (mouse click on the decrement button) event handler.

```
virtual bool  OnClickDec()
```

Returned value

true if the event has been processed, otherwise false.

OnChangeValue

The control "ChangeValue" event handler.

```
virtual bool  OnChangeValue()
```

Returned value

true if the event has been processed, otherwise false.

CDialog

CDialog is class of the Dialog complex control.

Description

CDialog class is intended to combine the controls with different functions in the group.

Declaration

```
class CDialog : public CWndContainer
```

Title

```
#include <Controls\Dialog.mqh>
```

Class Methods

Create	
Create	Creates control
Chart event handlers	
OnEvent	Event handler of all chart events
Properties	
Caption	Gets/Sets the value of the "Caption" property
Add	
Add	Adds control to the client area
Dependent controls	
CreateWhiteBorder	Creates dependent control (white border)
CreateBackground	Creates dependent control (background)
CreateCaption	Creates dependent control (caption)
CreateButtonClose	Creates dependent control (close button)
CreateClientArea	Creates dependent control (client area)
Dependent controls event handlers	
OnClickCaption	"ClickCaption" event handler
OnClickButtonClose	"ClickButtonClose" event handler
Access to client area	
ClientAreaVisible	Sets a value indicating whether the client area is visible
ClientAreaLeft	Gets X coordinate of the upper-left corner of the control client area

ClientAreaTop	Gets Y coordinate of the upper-left corner of the control client area
ClientAreaRight	Gets X coordinate of the lower-right corner of the control client area
ClientAreaBottom	Gets Y coordinate of the lower-right corner of the control client area
ClientAreaWidth	Gets the client area width
ClientAreaHeight	Gets the client area height
Drag event handlers	
OnDialogDragStart	"DialogDragStart" event handler (virtual)
OnDialogDragProcess	"DialogDragProcess" event handler (virtual)
OnDialogDragEnd	"DialogDragEnd" event handler (virtual)

Create

Creates new CDialog control.

```
virtual bool  Create(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

Caption (Get method)

Gets the "Caption" property of the CDialog control.

```
string  MinValue()  const
```

Returned value

The "Caption" property.

Caption (Set method)

Sets the "Caption" property of the CDialog control.

```
bool  Caption(
   const string  text      // text
   )
```

Parameters

text

[in] New value of "Caption" property.

Returned value

true if successful, otherwise false.

Add

Adds control to the client area by pointer.

```
bool  Add(
   CWnd   *control,        // pointer
   )
```

Parameters

control

[in] Pointer to control.

Returned value

true if successful, otherwise false.

Add

Adds control to the client area by reference.

```
bool  Add(
   CWnd   &control,        // reference
   )
```

Parameters

control

[in] Reference to control.

Returned value

true if successful, otherwise false.

CreateWhiteBorder

Creates dependent control (white border).

```
virtual bool  CreateWhiteBorder()
```

Returned value

true if successful, otherwise false.

CreateBackground

Creates dependent control (background).

```
virtual bool  CreateBackground()
```

Returned value

true if successful, otherwise false.

CreateCaption

Creates dependent control (caption).

```
virtual bool  CreateCaption()
```

Returned value

true if successful, otherwise false.

CreateButtonClose

Creates dependent control (close button)

```
virtual bool  CreateButtonClose()
```

Returned value

true if successful, otherwise false.

CreateClientArea

Creates dependent control (client area).

```
virtual bool  CreateClientArea()
```

Returned value

true if successful, otherwise false.

OnClickCaption

The control "ClickCaption" event handler.

```
virtual bool  OnClickCaption()
```

Returned value

true if successful, otherwise false.

OnClickButtonClose

The control "ClickButtonClose" event handler.

```
virtual bool  OnClickButtonClose()
```

Returned value

true if successful, otherwise false.

ClientAreaVisible

Sets a flag indicating whether the client area is visible.

```
bool  ClientAreaVisible(
   const bool  visible      // visibility flag
   )
```

Parameters

visible

[in] Visibility flag.

Returned value

true if successful, otherwise false.

ClientAreaLeft

Gets X coordinate of the upper-left corner of the control client area.

```
int  ClientAreaLeft()
```

Returned value

The X coordinate of the upper-left corner of the control client area.

ClientAreaTop

Gets Y coordinate of the upper-left corner of the control client area.

```
int  ClientAreaTop()
```

Returned value

The Y coordinate of the upper-left corner of the control client area.

ClientAreaRight

Gets X coordinate of the lower-right corner of the control client area.

```
int  ClientAreaTop()
```

Returned value

The X coordinate of the lower-right corner of the control client area.

ClientAreaBottom

Gets Y coordinate of the lower-right corner of the control client area.

```
int  ClientAreaBottom()
```

Returned value

The Y coordinate of the lower-right corner of the control client area.

ClientAreaWidth

Gets the width of the control client area.

```
int  ClientAreaWidth()
```

Returned value

The width of the client area.

ClientAreaHeight

Gets the height of the control client area.

```
int  ClientAreaHeight()
```

Returned value

The height of the control client area.

OnDialogDragStart

The control "DialogDragStart" event handler.

```
virtual bool  OnDialogDragStart()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "DialogDragStart" event occurs at start of the drag of the control.

OnDialogDragProcess

The control "DialogDragProcess" event handler.

```
virtual bool  OnDialogDragProcess()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "DialogDragProcess" event occurs when the control is dragged.

OnDialogDragEnd

The control "DialogDragEnd" event handler.

```
virtual bool  OnDialogDragEnd()
```

Returned value

true if the event has been processed, otherwise false.

Note

The "DialogDragEnd" event occurs at the end of the drag of the control.

CAppDialog

CAppDialog is a class of Application Dialog complex control (with dependent controls).

Description

CAppDialog class is intended to combine the controls with different functions in the group inside the MQL5 program.

Declaration

```
class CAppDialog : public CDialog
```

Title

```
#include <Controls\Dialog.mqh>
```

Class Methods

Create and destroy	
Create	Creates control
Destroy	Destroys control
Events processing	
OnEvent	Event handler of all chart events
Run	
Run	Runs control
Chart events processing	
ChartEvent	Event handler of all chart events
Settings	
Minimized	Sets a value indicating whether the control is minimized
Save/Load	
SaveIniFile	Saves the control state to file
LoadIniFile	Loads the control state from file
IniFileName	Sets the file name for loading/saving the control state
IniFileExt	Sets the file extension for loading/saving the control state
Initialization	
CreateCommon	Common initialization method

CreateExpert	Initialization method for working in Expert Advisors
CreateIndicator	Initialization method for working in indicators
Dependent controls	
CreateButtonMinMax	Creates dependent controls (minimize/ maximize buttons)
Dependent controls event handlers	
OnClickButtonClose	"ClickButtonClose" event handler (virtual)
OnClickButtonMinMax	"ClickButtonMinMax" event handler (virtual)
External events	
OnAnotherApplicationClose	Event handler of external events (virtual)
Methods	
Rebound	Sets new coordinates of the control using CRect class coordinates
Minimize	Shows the control in the minimized state
Maximize	Shows the control in the maximized (restored) state
CreateInstanceId	Creates an unique Id for the names of the control objects
ProgramName	Gets the name of MQL5 program
SubwinOff	Get the Y offset of the control subwindow

Create

Creates new CAppDialog control.

```
virtual bool  Create(
   const long     chart,      // chart ID
   const string   name,       // name
   const int      subwin,     // chart subwindow
   const int      x1,         // x1 coordinate
   const int      y1,         // y1 coordinate
   const int      x2,         // x2 coordinate
   const int      y2          // y2 coordinate
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

Destroy

Destroys control.

```
virtual bool  Destroy()
```

Returned value

true if successful, otherwise false.

OnEvent

Chart event handler.

```
virtual bool  OnEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

Run

Runs control.

```
bool  Run()
```

Returned value

true if successful, otherwise false.

ChartEvent

Chart event handler.

```
virtual bool  ChartEvent(
   const int      id,          // ID
   const long&    lparam,      // event parameter of long type
   const double&  dparam,      // event parameter of double type
   const string&  sparam       // event parameter of string type
   )
```

Parameters

id

[in] Event ID.

lparam

[in] Event parameter of long type, passed by reference.

dparam

[in] Event parameter of double type, passed by reference.

sparam

[in] Event parameter of string type, passed by reference.

Returned value

true - if the event has been processed, otherwise false.

Minimized

Sets the value of "Minimized" property of the control.

```
bool  Minimized(
   const bool  flag      // state
   )
```

Parameters

flag

[in] New state.

Returned value

true if successful, otherwise false.

SaveIniFile

Saves the control state to file.

```
void  SaveIniFile()
```

Returned value

true if successful, otherwise false.

LoadIniFile

Loads the control state from file.

```
void  LoadIniFile()
```

Returned value

true if successful, otherwise false.

IniFileName

Sets the file name for loading/saving the control state.

```
virtual string  IniFileName()  const
```

Returned value

File name for loading/saving of the control state.

Note

The file name include the name of the Expert Advisor/indicator and working symbol, on which MQL5 program is launched.

IniFileExt

Sets the file extension for loading/saving the control state.

```
virtual string  IniFileExt()  const
```

Returned value

File extension, used for loading/saving of the control state.

CreateCommon

Common initialization method.

```
bool  CreateCommon(
   const long     chart,       // chart ID
   const string   name,        // name
   const int      subwin,      // chart subwindow
   )
```

Parameters

chart

[in] chart ID.

name

[in] Unique name of the control.

subwin

[in] Chart subwindow.

Returned value

true if successful, otherwise false.

CreateExpert

Initialization method for working in Expert Advisors.

```
bool  CreateExpert(
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

CreateIndicator

Initialization method for working in indicators.

```
bool  CreateIndicator(
   const int      x1,          // x1 coordinate
   const int      y1,          // y1 coordinate
   const int      x2,          // x2 coordinate
   const int      y2           // y2 coordinate
   )
```

Parameters

x1

[in] X coordinate of the upper left corner.

y1

[in] Y coordinate of the upper left corner.

x2

[in] X coordinate of the lower right corner.

y2

[in] Y coordinate of the lower right corner.

Returned value

true if successful, otherwise false.

CreateButtonMinMax

Creates dependent controls (minimize/maximize buttons).

```
virtual void  CreateButtonMinMax()
```

Returned value

None.

OnClickButtonClose

The control "ClickButtonClose" (mouse click on close button) event handler.

```
virtual void  OnClickButtonClose()
```

Returned value

None.

OnClickButtonMinMax

The control "ClickButtonMinMax" (mouse click on minimize/maximize button) event handler.

```
virtual void  OnClickButtonClose()
```

Returned value

None.

OnAnotherApplicationClose

Event handler of external events.

```
virtual void  OnAnotherApplicationClose()
```

Returned value

None.

Rebound

Sets new coordinates of the control using CRect class coordinates.

```
bool  Rebound(
   const & CRect  rect      // CRect class
   )
```

Returned value

true if successful, otherwise false.

Minimize

Shows the control in the minimized state.

```
virtual void  Minimize()
```

Returned value

true if successful, otherwise false.

Maximize

Shows the control in the maximized (restored) state.

```
virtual void  Maximize()
```

Returned value

true if successful, otherwise false.

CreateInstanceId

Creates an unique Id for the names of the control objects.

```
string  CreateInstanceId()
```

Returned value

Prefix for the object names.

ProgramName

Gets the name of the MQL5 program.

```
string  ProgramName()
```

Returned value

Name of the MQL5 program.

SubwinOff

Get the Y offset of the control subwindow.

```
void  SubwinOff()
```

Returned value

None.

Paso de la versión MQL4

El lenguaje MQL5 es un lenguaje desarrollado de su antecesor MQL4, en el cual está escrita una gran cantidad de indicadores, scripts y Asesores Expertos. A pesar de que el lenguaje nuevo es compatible al máximo con la versión anterior, existen unas diferencias entre ellos. Y al pasar los programas hay que saber estas diferencias.

En este apartado está reunida la información cuyo objetivo es facilitar a los programadores que conocen bien Mql4 la adaptación de sus códigos al nuevo lenguaje MQL5.

Ante todo cabe señalar que:

- No hay funciones start(), init() y deinit();
- Cantidad de buffers de indicador no está limitada;
- dll se carga una vez cargado un Asesor Experto (o cualquier otro programa mql5);
- Comprobación reducida de condiciones lógicas;
- Al salir fuera de los límites de un array la ejecución en curso se detiene (de una forma crítica - con mensaje del error);
- Prioridad de operaciones igual que en C++;
- Funciona la conversión implícita de tipos (incluso de cadena a un número);
- Variables locales no se inicializan automáticamente (salvo las cadenas);
- Arrays locales comunes se borran automáticamente.

Funciones especiales init, start y deinit

En el lenguaje MQL4 había sólo tres funciones predefinidas que podían estar presentes en el código de un indicador, Asesor Experto o script (sin tener en cuenta los archivos incluidos *.mqh y archivos de bibliotecas). En MQL5 no hay estas funciones, pero hay sus análogos. En la tabla de abajo se muestra una correspondencia aproximada de estas funciones.

MQL4	MQL5
init	OnInit
start	OnStart
deinit	OnDeinit

Las funciones OnInit y OnDeinit desempeñan el mismo papel que las funciones init y deinit en MQL4, es decir, están predestinadas para colocar el código que debe ejecutarse con la inicialización y deinicialización de los programas mql5. Se puede simplemente renombrar estas funciones de una manera correspondiente, o bien dejarlas tal como están pero hay que añadir la llamada a éstas en los sitios correspondientes.

Ejemplo:

```
void OnInit()
  {
//--- llamamos a la función al inicializar
   init();
```

```
  }
void OnDeinit(const int reason)
  {
//--- llamamos a la función al deinicializar
   deinit();
//---
  }
```

La función start se cambia por OnStart sólo en los scripts, para un Asesor Experto o un indicador hace falta renombrarla con OnTick y OnCalculate respectivamente. El código que tiene que ser ejecutado durante el funcionamiento de un programa mql5, hay que colocarlo precisamente en estas tres funciones:

programa mql5	función principal
script	OnStart
indicador	OnCalculate
Asesor Experto	OnTick

Si el código del indicador o script no contiene la función principal o el nombre de ésta se diferencia del requerido, la llamada a esta función no se realiza. Es decir, si en el código principal de un script no hay función OnStart, este código va a ser compilado como un Asesor Experto.

Si en el código de un indicador falta la función OnCalculate, la compilación de este indicador no es posible.

Variables predefinidas

En MQL5 no hay variables predefinidas como Ask, Bid, Bars. Las variables Digits y Point se han cambiado un poco tal como se muestra en la tabla.

MQL4	MQL5
Digits	_Digits
Point	_Point
	_LastError
	_Period
	_Symbol
	_StopFlag
	_UninitReason

Acceso a series temporales

En MQL5 no hay series temporales predefinidas Open[], High[], Low[], Close[], Volume[] y Time[].

Ahora se puede establecer independientemente la profundidad necesaria de series temporales a través de las correspondientes funciones para acceder a las series temporales.

Asesores Expertos

En MQL5 los Asesores Expertos no requieren la presencia obligatoria de la función-manejador del evento de llegada de un nuevo tick OnTick, como ha sido en MQL4 (en MQL4 la función start se invoca para ser ejecutada cuando llega un nuevo tick), porque ahora en MQL5 los Asesores Expertos pueden contener las funciones-manejadores predefinidas de varios tipos de eventos:

- OnTick – recepción de un nuevo tick;
- OnTimer – evento de temporizador;
- OnTrade - evento comercial;
- OnChartEvent – evento de introducción por el teclado o ratón, evento de movimiento de un objeto gráfico, evento del fin de edición del texto en la casilla de introducción del objeto LabelEdit;
- OnBookEvent – evento del cambio del estado de profundidad de mercado (Depth of Market).

Indicadores personalizados

En MQL4 la cantidad de los buffers de indicador está limitada y no puede superar 8. En MQL5 tal limitación no existe pero hay que recordar que cada búfer de indicadores requiere un cierto volumen de memoria operativa para su ubicación en el terminal, por eso a pesar de todo no hace falta abusar de esta posibilidad.

Además, en MQL4 había sólo 6 tipos de representar gráficamente un indicador personalizado, en MQL5 ahora hay 18 estilos de dibujo. Y aunque los nombres de los tipos de dibujar no se hayan cambiado, la ideología de representación gráfica de los indicadores se ha cambiado considerablemente.

La dirección de la indexación en los buffers de indicadores también se diferencia. Por defecto, en MQL5 todos los búfers se comportan como los arrays comunes, es decir, un elemento con el índice 0 es el más antiguo de la historia, con el aumento del índice nos movemos de los datos más antiguos a los recientes.

La única función para trabajar con indicadores personalizados que se ha conservado de MQL4 es la SetIndexBuffer. Pero su llamada también se ha cambiado, ahora tenemos que especificar tipo de datos que van a guardarse en el array vinculado con el búfer de indicadores.

A parte de eso, las propiedades de los indicadores personalizados se han cambiado y se han extendido. Han sido agregadas nuevas funciones de acceso a series temporales, por eso hay que reconsiderar completamente todo el algoritmo de cálculo.

Objetos gráficos

El número de objetos gráficos en MQL5 se ha incrementado de una manera significativa, sin embargo, ha aparecido una restricción, es que no se puede usar las funciones de trabajo con objetos gráficos en los indicadores personalizados.

Además, el posicionamiento de objetos gráficos en el tiempo ahora es posible con una precisión de hasta un segundo en un gráfico de cualquier período, ahora no se realiza el redondeo de puntos de

anclaje de objetos gráficos con precisión de tiempo de apertura del bar en el actual gráfico de precios.

Para los objetos Arrow, Text y Label ha aparecido la posibilidad de indicar el modo de anclaje, y para los objetos Label, Button, Chart, Bitmap Label y Edit se puede determinar la esquina del gráfico a la que está anclado el objeto.