

# Справочник MQL4

Содержание	стр.
<b>1 ОСНОВЫ ЯЗЫКА.....</b>	<b>10</b>
1.1 СИНТАКСИС .....	10
1.1.1 Комментарии.....	10
1.1.2 Идентификаторы.....	11
1.1.3 Резервированные слова.....	11
1.2 ТИПЫ ДАННЫХ.....	12
1.2.1 Приведение типов.....	12
1.2.2 Целые константы (int).....	12
1.2.3 Символьные константы.....	13
1.2.4 Логические константы (bool).....	14
1.2.5 Константы с плавающей точкой (double).....	14
1.2.6 Строковые константы (string).....	14
1.2.7 Цветовые константы (color).....	15
1.2.8 Константы даты и времени (datetime).....	16
1.3 ОПЕРАЦИИ И ВЫРАЖЕНИЯ .....	16
1.3.1 Выражения .....	17
1.3.2 Арифметические операции.....	17
1.3.3 Операция присваивания .....	18
1.3.4 Операции отношения.....	19
1.3.5 Логические операции.....	19
1.3.6 Побитовые операции.....	20
1.3.7 Другие операции.....	21
1.3.8 Приоритеты и порядок выполнения операций .....	22
1.4 ОПЕРАТОРЫ .....	23
1.4.1 Составной оператор.....	24
1.4.2 Оператор-выражение .....	24
1.4.3 Оператор завершения break.....	25
1.4.4 Оператор продолжения continue .....	25
1.4.5 Оператор возврата return.....	26
1.4.6 Условный оператор if-else .....	26
1.4.7 Оператор-переключатель switch.....	28
1.4.8 Оператор цикла while.....	29
1.4.9 Оператор цикла for.....	29
1.5 ФУНКЦИИ .....	31
1.5.1 Вызов функции .....	32
1.5.2 Специальные функции.....	33
1.6 ПЕРЕМЕННЫЕ .....	34
1.6.1 Локальные переменные.....	35
1.6.2 Формальные параметры .....	36
1.6.3 Статические переменные .....	38
1.6.4 Глобальные переменные.....	39
1.6.5 Внешние переменные .....	40
1.6.6 Инициализация переменных.....	41
1.6.7 Описание внешних функций.....	41
1.7 ПРЕПРОЦЕССОР .....	43
1.7.1 Объявление константы .....	43
1.7.2 Специальные параметры.....	44
1.7.3 Включение файлов.....	45
1.7.4 Импорт функций.....	46
<b>2 СТАНДАРТНЫЕ КОНСТАНТЫ .....</b>	<b>49</b>
2.1 ТАЙМСЕРИИ .....	49

2.2 ПЕРИОДЫ ГРАФИКОВ .....	49
2.3 ТОРГОВЫЕ ОПЕРАЦИИ .....	50
2.4 ЦЕНОВЫЕ КОНСТАНТЫ .....	50
2.5 MARKETINFO .....	51
2.6 СТИЛИ РИСОВАНИЯ .....	53
2.7 КОДЫ СТРЕЛОК .....	54
2.8 WINGDINGS .....	55
2.9 НАБОР WEB-ЦВЕТОВ .....	57
2.10 ЛИНИИ ИНДИКАТОРОВ .....	58
2.11 ИСНМОКУ KINKO NYO .....	59
2.12 МЕТОДЫ СКОЛЬЗЯЩИХ .....	59
2.13 MESSAGEBOX .....	60
2.14 ТИПЫ ОБЪЕКТОВ .....	62
2.15 СВОЙСТВА ОБЪЕКТОВ .....	66
2.16 ВИДИМОСТЬ ОБЪЕКТОВ .....	68
2.17 ПРИЧИНЫ ДЕИНИЦИАЛИЗАЦИИ .....	71
2.18 СПЕЦИАЛЬНЫЕ КОНСТАНТЫ .....	72
2.18 КОДЫ ОШИБОК .....	72
<b>3 ВЫПОЛНЕНИЕ ПРОГРАММ .....</b>	<b>80</b>
3.1 ВЫПОЛНЕНИЕ ПРОГРАММ .....	80
3.2 ВЫЗОВ ИМПОРТИРУЕМЫХ ФУНКЦИЙ .....	82
3.3 ОШИБКИ ВЫПОЛНЕНИЯ .....	84
<b>4 ПРЕДОПРЕДЕЛЕННЫЕ ПЕРЕМЕННЫЕ .....</b>	<b>100</b>
4.1 ASK .....	100
4.2 BARS .....	100
4.3 BID .....	101
4.4 CLOSE .....	101
4.5 DIGITS .....	101
4.6 HIGH .....	102
4.7 LOW .....	102
4.8 OPEN .....	103
4.9 POINT .....	104
4.10 TIME .....	104
4.11 VOLUME .....	105
<b>5 ИНФОРМАЦИЯ О СЧЕТЕ .....</b>	<b>106</b>
5.1 ACCOUNTBALANCE .....	106
5.2 ACCOUNTCREDIT .....	106
5.3 ACCOUNTCOMPANY .....	106
5.4 ACCOUNTCURRENCY .....	106
5.5 ACCOUNTEQUITY .....	106
5.6 ACCOUNTFREEMARGIN .....	107
5.7 ACCOUNTFREEMARGINCHECK .....	107
5.8 ACCOUNTFREEMARGINMODE .....	107
5.9 ACCOUNTLEVERAGE .....	108
5.10 ACCOUNTMARGIN .....	108
5.11 ACCOUNTNAME .....	108
5.12 ACCOUNTNUMBER .....	108
5.13 ACCOUNTPROFIT .....	109
5.14 ACCOUNTSERVER .....	109
5.15 ACCOUNTSTOPOUTLEVEL .....	109
5.16 ACCOUNTSTOPOUTMODE .....	109
<b>6 ОПЕРАЦИИ С МАССИВАМИ .....</b>	<b>110</b>
6.1 ARRAYBSEARCH .....	110
6.2 ARRAYCOPY .....	111
6.3 ARRAYCOPYRATES .....	112
6.4 ARRAYCOPYSERIES .....	113
6.5 ARRAYDIMENSION .....	114
6.6 ARRAYGETASERIES .....	114
6.7 ARRAYINITIALIZE .....	115

6.8 ARRAYISERIES.....	115
6.9 ARRAYMAXIMUM.....	116
6.10 ARRAYMINIMUM.....	116
6.11 ARRAYRANGE.....	116
6.12 ARRAYRESIZE.....	117
6.13 ARRAYSETASERIES.....	117
6.14 ARRAYSIZE.....	118
6.15 ARRAYSORT.....	118
<b>7 ПРОВЕРКА СОСТОЯНИЯ.....</b>	<b>120</b>
7.1 GETLASTERROR.....	120
7.2 ISCONNECTED.....	120
7.3 ISDEMO.....	120
7.4 ISDLLSALLOWED.....	121
7.5 ISEXPERTENABLED.....	121
7.6 ISLIBRARIESALLOWED.....	121
7.7 ISOPTIMIZATION.....	122
7.8 ISSTOPPED.....	122
7.9 ISTESTING.....	122
7.10 ISTRADEALLOWED.....	122
7.11 ISTRADECONTEXTBUSY.....	123
7.12 ISVISUALMODE.....	123
7.13 UNINITIALIZEREASON.....	123
<b>8 КЛИЕНТСКИЙ ТЕРМИНАЛ.....</b>	<b>125</b>
8.1 TERMINALCOMPANY.....	125
8.2 TERMINALNAME.....	125
8.3 TERMINALPATH.....	125
<b>9 ОБЩИЕ ФУНКЦИИ.....</b>	<b>126</b>
9.1 ALERT.....	126
9.2 COMMENT.....	126
9.3 GETTICKCOUNT.....	127
9.4 MARKETINFO.....	127
9.5 MESSAGEBOX.....	128
9.6 PLAYSOUND.....	129
9.7 PRINT.....	129
9.8 SENDFTP.....	130
9.9 SENDMAIL.....	130
9.10 SLEEP.....	131
<b>10 ПРЕОБРАЗОВАНИЯ ДАННЫХ.....</b>	<b>132</b>
10.1 CHARTOSTR.....	132
10.2 DOUBLETOSTR.....	132
10.3 NORMALIZEDOUBLE.....	132
10.4 STRTODOUBLE.....	133
10.5 STRTOINTEGER.....	133
10.6 STRTOTYPE.....	133
10.7 TIMETOSTR.....	134
<b>11 ПОЛЬЗОВАТЕЛЬСКИЕ ИНДИКАТОРЫ.....</b>	<b>135</b>
11.1 INDICATORBUFFERS.....	135
11.2 INDICATORCOUNTED.....	136
11.3 INDICATORDIGITS.....	136
11.4 INDICATORSHORTNAME.....	137
11.5 SETINDEXARROW.....	138
11.6 SETINDEXBUFFER.....	138
11.7 SETINDEXDRAWBEGIN.....	139
11.8 SETINDEXEMPTYVALUE.....	140
11.9 SETINDEXLABEL.....	140
11.10 SETINDEXSHIFT.....	142
11.11 SETINDEXSTYLE.....	143
11.12 SETLEVELSTYLE.....	143

11.13 SETLEVELVALUE .....	144
<b>12 ДАТА И ВРЕМЯ.....</b>	<b>145</b>
12.1 DAY .....	145
12.2 DAYOFWEEK .....	145
12.3 DAYOFYEAR .....	145
12.4 HOUR .....	145
12.5 MINUTE .....	146
12.6 MONTH.....	146
12.7 SECONDS .....	146
12.8 TIMECURRENT .....	147
12.9 TIMEDAY .....	147
12.10 TIMEDAYOFWEEK.....	147
12.11 TIMEDAYOFYEAR .....	147
12.12 TIMEHOUR.....	148
12.13 TIMELOCAL .....	148
12.14 TIMEMINUTE .....	148
12.15 TIMEMONTH.....	149
12.16 TIMESECONDS .....	149
12.17 TIMEYEAR.....	149
12.18 YEAR.....	150
<b>13 ФАЙЛОВЫЕ ОПЕРАЦИИ.....</b>	<b>152</b>
13.1 FILECLOSE.....	152
13.2 FILEDELETE.....	152
13.3 FILEFLUSH.....	153
13.4 FILEISENDING.....	153
13.5 FILEISLINEENDING .....	154
13.6 FILEOPEN .....	154
13.7 FILEOPENHISTORY .....	155
13.8 FILEREADARRAY .....	156
13.9 FILEREADDOUBLE .....	157
13.10 FILEREADINTEGER.....	157
13.11 FILEREADNUMBER .....	158
13.12 FILEREADSTRING.....	158
13.13 FILESEEK.....	159
13.14 FILESIZE .....	159
13.15 FILETELL.....	160
13.16 FILEWRITE .....	160
13.17 FILEWRITEARRAY .....	161
13.18 FILEWRITEDOUBLE.....	162
13.19 FILEWRITEINTEGER .....	163
13.20 FILEWRITESTRING .....	163
<b>14 ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ.....</b>	<b>165</b>
14.1 GLOBALVARIABLECHECK .....	165
14.2 GLOBALVARIABLEDEL .....	165
14.3 GLOBALVARIABLEGET .....	166
14.4 GLOBALVARIABLENAME .....	166
14.5 GLOBALVARIABLESET .....	166
14.6 GLOBALVARIABLESETONCONDITION.....	167
14.7 GLOBALVARIABLESDELETEALL.....	168
14.8 GLOBALVARIABLESTOTAL.....	168
<b>15 МАТЕМАТИЧЕСКИЕ ФУНКЦИИ.....</b>	<b>169</b>
15.1 MATHABS .....	169
15.2 MATHARCCOS .....	169
15.3 MATHARCSIN .....	169
15.4 MATHARCTAN.....	170
15.5 MATHCEIL.....	170
15.6 MATHCOS .....	170
15.7 MATHEXP.....	171
15.8 MATHFLOOR .....	171

15.8 MATHLOG .....	172
15.9 MATHMAX .....	172
15.10 MATHMIN .....	172
15.11 MATHMOD .....	173
15.12 MATHPOW .....	173
15.13 MATHRAND .....	173
15.14 MATHROUND .....	174
15.15 MATHSIN .....	174
15.16 MATHSQRT .....	174
15.17 MATHSRAND .....	175
15.18 MATHTAN .....	175
<b>16 ГРАФИЧЕСКИЕ ОБЪЕКТЫ.....</b>	<b>177</b>
16.1 OBJECTCREATE .....	177
16.2 OBJECTDELETE .....	178
16.3 OBJECTDESCRIPTION.....	178
16.4 OBJECTFIND .....	179
16.5 OBJECTGET .....	179
16.6 OBJECTGETFIBODESCRIPTION.....	180
16.7 OBJECTGETSHIFTBYVALUE .....	180
16.8 OBJECTGETVALUEBYSHIFT .....	181
16.9 OBJECTMOVE .....	181
16.10 OBJECTNAME .....	182
16.11 OBJECTSDELETEALL.....	182
16.12 OBJECTSET .....	183
16.13 OBJECTSETFIBODESCRIPTION.....	183
16.14 OBJECTSETTEXT .....	184
16.15 OBJECTSTOTAL.....	184
16.16 OBJECTTYPE .....	185
<b>17 СТРОКОВЫЕ ФУНКЦИИ .....</b>	<b>186</b>
17.1 STRINGCONCATENATE .....	186
17.2 STRINGFIND.....	186
17.3 STRINGGETCHAR.....	187
17.4 STRINGLEN.....	187
17.5 STRINGSETCHAR .....	187
17.6 STRINGSUBSTR .....	188
17.7 STRINGTRIMLEFT .....	188
17.8 STRINGTRIMRIGHT .....	188
<b>18 ТЕХНИЧЕСКИЕ ИНДИКАТОРЫ .....</b>	<b>190</b>
18.1 IAC .....	190
18.2 IAD.....	191
18.3 IALLIGATOR.....	191
18.4 IADX .....	192
18.5 IATR.....	193
18.6 IAO .....	193
18.7 IBEARSPower .....	194
18.8 IBANDS.....	194
18.9 IBANDSONARRAY.....	195
18.10 IBULLSPower .....	195
18.11 ICCI .....	196
18.12 ICCIONARRAY .....	197
18.13 ICUSTOM .....	197
18.14 IDEMARKER .....	198
18.15 IENVELOPES.....	198
18.16 IENVELOPESONARRAY.....	199
18.17 IFORCE .....	200
18.18 IFRACTALS.....	201
18.19 IGATOR.....	201
18.20 ICHIMOKU.....	202
18.21 IBWMFI.....	203
18.22 IMOMENTUM.....	204

18.23 iMOMENTUMONARRAY .....	204
18.24 iMFI .....	205
18.25 iMA .....	205
18.26 iMAONARRAY .....	206
18.27 iOsMA .....	207
18.28 iMACD .....	208
18.29 iOBV .....	209
18.30 iSAR .....	209
18.31 iRSI .....	210
18.32 iRSIONARRAY .....	210
18.33 iRVI .....	211
18.34 iSTDDEV .....	211
18.35 iSTDDEVONARRAY .....	212
18.36 iSTOCHASTIC .....	213
18.37 iWPR .....	214
<b>19 ДОСТУП К ТАЙМСЕРИЯМ.....</b>	<b>215</b>
19.1 iBARS .....	215
19.2 iBARSHIFT .....	215
19.3 iCLOSE .....	216
19.4 iHIGH .....	217
19.5 iHIGHEST .....	217
19.6 iLOW .....	218
19.7 iLOWEST .....	219
19.8 iOPEN .....	219
19.9 iTIME .....	220
19.10 iVOLUME .....	221
<b>20 ТОРГОВЫЕ ФУНКЦИИ .....</b>	<b>222</b>
20.1 ОШИБКИ ИСПОЛНЕНИЯ .....	222
20.2 ORDERCLOSE .....	230
20.3 ORDERCLOSEBY .....	230
20.4 ORDERCLOSEPRICE .....	231
20.5 ORDERCLOSETIME .....	231
20.6 ORDERCOMMENT .....	232
20.7 ORDERCOMMISSION .....	232
20.8 ORDERDELETE .....	232
20.9 ORDEREXPIRATION .....	233
20.10 ORDERLOTS .....	233
20.11 ORDERMAGICNUMBER .....	233
20.12 ORDERMODIFY .....	233
20.13 ORDEROPENPRICE .....	235
20.14 ORDEROPENTIME .....	235
20.15 ORDERPRINT .....	235
20.16 ORDERPROFIT .....	236
20.17 ORDERSELECT .....	236
20.18 ORDERSEND .....	237
20.19 ORDERSHISTORYTOTAL .....	239
20.20 ORDERSTOPLOSS .....	240
20.21 ORDERSTOTAL .....	240
20.22 ORDERSWAP .....	240
20.23 ORDERSYMBOL .....	241
20.24 ORDERTAKEPROFIT .....	241
20.25 ORDERTICKET .....	241
20.26 ORDERTYPE .....	241
<b>21 ОПЕРАЦИИ С ГРАФИКАМИ.....</b>	<b>243</b>
21.1 HIDETESTINDICATORS .....	243
21.2 PERIOD .....	243
21.3 REFRESHRATES .....	243
21.4 SYMBOL .....	244
21.5 WINDOWBARSPERCHART .....	244
21.6 WINDOWEXPERTNAME .....	245

21.7 WINDOWFIND .....	245
21.8 WINDOWFIRSTVISIBLEBAR.....	245
21.9 WINDOWHANDLE .....	245
21.10 WINDOWISVISIBLE .....	246
21.11 WINDOWONDROPPED .....	246
21.13 WINDOWPRICEMAX.....	246
21.14 WINDOWPRICEMIN .....	247
21.15 WINDOWPRICEONDROPPED .....	247
21.16 WINDOWREDRAW .....	248
21.17 WINDOWSCREENSHOT .....	248
21.18 WINDOWTIMEONDROPPED .....	248
21.19 WINDOWSTOTAL .....	249
21.20 WINDOWXONDROPPED.....	249
21.21 WINDOWYONDROPPED.....	249
<b>22 УСТАРЕВШИЕ ФУНКЦИИ .....</b>	<b>250</b>

MetaQuotes Language 4 (MQL4) - новый встроенный язык программирования торговых стратегий. Этот язык позволяет писать собственные программы-эксперты (Expert Advisors), автоматизирующие управление торговыми процессами и идеально подходящие для реализации собственных торговых стратегий. Кроме того, на MQL4 можно создавать собственные технические индикаторы (Custom Indicators), скрипты (Scripts) и библиотеки функций (Libraries).

В состав MQL4 включено большое количество функций, необходимых для анализа текущих и прошедших ранее котировок, встроены основные индикаторы и функции по управлению торговыми позициями и контролю над ними.

Для написания кода программы используется текстовый редактор экспертов MetaEditor 4, выделяющий цветом различные конструкции языка MQL4, что позволяет пользователю лучше ориентироваться в тексте экспертной системы. В качестве справочной системы по языку MQL4 используется словарь - MetaQuotes Language Dictionary. Краткий справочник содержит разбитые на категории функции, операции, зарезервированные слова, другие конструкции языка и позволяет узнать описание каждого используемого элемента, входящего в язык.

Программы, написанные на MetaQuotes Language 4, имеют различные свойства и предназначение:

- **Советник** (Expert Advisor) - это механическая торговая система (МТС), имеющая привязку к определенному графику. Советник запускается на выполнение с каждым поступающим тиком по данному инструменту. Советник не будет запущен для вновь поступившего тика, если в этот момент советник обрабатывает предыдущий тик (то есть, советник еще не закончил свою работу). Советник может не только работать в режиме информирования о возможности совершить сделки, но и автоматически совершать сделки на торговом счете, направляя их прямо на торговый сервер. Как и в большинстве информационных систем, в терминале поддерживается тестирование стратегий на исторических данных с отображением на графиках точек входа в торговые позиции и выхода из них. Советники хранятся в директории `каталог_терминала\experts`
- **Пользовательский индикатор** (Custom Indicator) - технический индикатор, самостоятельно написанный пользователем в дополнение к индикаторам, уже интегрированным в клиентский терминал. Пользовательские индикаторы, также как и встроенные, не могут автоматически торговать и предназначены только для

реализации аналитических функций. Пользовательские индикаторы хранятся в директории **каталог\_терминала\experts\indicators**

- **Скрипт (Script)** - программа, предназначенная для одноразового выполнения каких-либо действий. В отличие от экспертов, скрипты запускаются не потиково, а по запросу. Скрипты хранятся в директории **каталог\_терминала\experts\scripts**
- **Библиотека (Library)** - библиотека пользовательских функций, предназначенная для хранения и распространения часто используемых блоков пользовательских программ. Библиотеки не могут самостоятельно запускаться на выполнение. Библиотеки рекомендуется хранить в директории **каталог\_терминала\experts\libraries**
- **Включаемый файл (Included file)** - исходный текст часто используемых блоков пользовательских программ. Такие файлы могут включаться в исходные тексты экспертов, скриптов, пользовательских индикаторов и библиотек на этапе компиляции. Использование включаемых файлов более предпочтительно, чем использование библиотек, из-за дополнительных накладных расходов при вызове библиотечных функций. Включаемые файлы рекомендуется хранить в директории **каталог\_терминала\experts\include**

# **1 Основы языка**

Язык MetaQuotes Language 4 (MQL4) - новый встроенный язык программирования торговых стратегий. Этот язык позволяет писать собственные программы-эксперты (Expert Advisors), автоматизирующие управление торговыми процессами и идеально подходящие для реализации собственных торговых стратегий. Кроме того, на MQL4 можно создавать собственные технические индикаторы (Custom Indicators), скрипты (Scripts) и библиотеки функций (Libraries).

## **1.1 Синтаксис**

Синтаксис языка программирования торговых стратегий MQL4 синтаксически очень похож язык программирования Си, за исключением некоторых возможностей:

- отсутствует адресная арифметика;
- отсутствует оператор do ... while;
- отсутствует оператор goto ...;
- отсутствует операция [условие]?[выражение 1]:[выражение 2];
- отсутствуют сложные типы данных (структуры);
- невозможны сложные присваивания. Например, val1=val2=0; arr[i++]=val; cond=(cnt=OrdersTotal)>0; и т.п.;
- вычисление логического выражения производится до конца и не прерывается досрочно

### **1.1.1 Комментарии**

Многострочные комментарии начинаются парой символов /\* и заканчиваются парой \*/. Данные комментарии не могут быть вложенными. Однострочные комментарии начинаются парой символов //, заканчиваются символом новой строки и могут быть вложены в многострочные комментарии. Комментарии разрешены везде, где возможны пробелы, и допускают любое число пробелов.

#### **Примеры:**

```
// Однострочный комментарий
/* Многострочный
   // Вложенный однострочный комментарий
   комментарий
```

\*/

### 1.1.2 Идентификаторы

Идентификаторы используются в качестве имен для переменных и функций. Длина идентификатора не может превышать 31 знак.

Допустимые символы: цифры 0-9, латинские прописные и строчные буквы a - z и A - Z, распознаваемые как разные символы, символ подчеркивания (\_). Первый символ не может быть цифрой. Идентификатор не должен совпадать с зарезервированным словом.

#### Примеры:

```
NAME1 name1 Total_5 Paper
```

### 1.1.3 Зарезервированные слова

Перечисленные ниже идентификаторы фиксируются как зарезервированные слова, каждому из которых соответствует определенное действие, и в другом смысле не могут использоваться:

Типы данных	Классы памяти	Операторы	Прочие
bool	extern	break	false
color	static	case	true
datetime		continue	
double		default	
int		else	
string		for	
void		if	
		return	
		switch	
		while	

## **1.2 Типы данных**

### **1.2.1 Приведение типов**

Любая программа оперирует данными. Данные могут быть различных типов в зависимости от назначения. Например, для доступа к элементам массива используются данные целочисленного типа. Ценовые данные имеют тип двойной точности с плавающей точкой. Это связано с тем, что в языке MQL 4 не предусмотрено специального типа для ценовых данных.

Данные разного типа обрабатываются с разной скоростью. Целочисленные данные обрабатываются быстрее всего. Для обработки данных двойной точности используется специальный сопроцессор. Однако из-за сложности внутреннего представления данных с плавающей точкой, они обрабатываются дольше, чем целочисленные. Дольше всего обрабатываются строковые данные. Это связано с динамическим распределением-перераспределением оперативной памяти компьютера.

Основные типы данных:

- целые (int)
- логические (bool)
- литералы (char)
- строки (string)
- с плавающей точкой (double)
- цвет (color)
- дата и время (datetime)

Типы *color* и *datetime* имеют смысл только для удобства представления и ввода параметров, задаваемых извне - из таблицы свойств советника или пользовательского индикатора (вкладка "Inputs"). Данные типов *color* и *datetime* представляются в виде целых чисел. Целые типы вместе с типами с плавающей точкой называются арифметическими (числовыми) типами.

В выражениях используется только неявное приведение типов.

### **1.2.2 Целые константы (int)**

Десятичные: цифры 0-9; первой цифрой не должен быть 0.

### Примеры:

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

Шестнадцатеричные: цифры 0-9, буквы a - f или A - F для значений 10-15; начинаются с 0x или 0X.

### Примеры:

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

Внутреннее представление - длинное целое число размером 4 байта. Целые константы могут принимать значения от -2147483648 до 2147483647. Если константа превышает указанный диапазон, то результат не определен.

### 1.2.3 Символьные константы

Любой одиночный символ, заключенный в одинарные кавычки, или шестнадцатеричный ASCII-код символа в виде '\x10' является символьной константой и имеет тип int.

Некоторые символы, например, одинарные кавычки ('), двойные кавычки ("), знак вопроса (?), обратная косая черта (\) и управляющие символы можно представлять комбинацией символов, начинающейся с обратной косой чертой(\), в соответствии с приводимой ниже таблицей:

новая строка (перевод строки)	NL (LF)	\n
горизонтальная табуляция	HT	\t
возврат каретки	CR	\r
обратная косая черта	\	\\
одинарная кавычка	'	'\''
двойная кавычка	"	'\"'
шестнадцатеричный ASCII-код	hh	\xhh

Если за обратной косой чертой следует символ, отличный от перечисленных, результат не определяется:

```
int a = 'A';  
int b = '$';  
int c = '©'; // код 0xA9  
int d = '\xAE'; // код символа ®
```

Внутреннее представление - длинное целое число размером 4 байта. Символьные константы могут принимать значения от 0 до 255. Если константа превышает указанный диапазон, то результат не определен

#### **1.2.4 Логические константы (bool)**

Логические константы имеют значение true (истина) или false (ложь), числовое представление которых 1 или 0 соответственно. Могут использоваться написания True, TRUE, False и FALSE.

##### **Примеры:**

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

Внутреннее представление - длинное целое число размером 4 байта. Логические константы могут принимать значения 0 и 1.

#### **1.2.5 Константы с плавающей точкой (double)**

Константы с плавающей точкой состоят из целой части, точки (.) и дробной части. Целая и дробная части представляют собой последовательности десятичных цифр.

##### **Примеры:**

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

Внутреннее представление - число двойной точности размером 8 байт. Пределы изменения от  $-1.7 * e^{-308}$  до  $1.7 * e^{308}$ . Точность обеспечивается не более, чем 15 значащими цифрами.

#### **1.2.6 Строковые константы (string)**

Строковая константа представляет собой последовательность символов кода ASCII, заключенную в двойные кавычки: "Character constant".

Строковая константа - это массив символов, заключенный в кавычки. Она имеет тип string. Если необходимо ввести в строку двойную кавычку ("), то перед ней надо поставить символ обратной косой черты (\). В строку могут быть введены любые специальные символьные константы, перед которыми стоит символ обратной косой черты (\). Длина строковой константы - от 0 до 255 символов. Если длина строковой константы превосходит максимальную, лишние символы справа отбрасываются, и компилятор выдает соответствующее предупреждение.

**Примеры:**

```
"This is a character string"  
"Это строковая константа"  
"Символ копирайта\t\xA9"  
"эта строка содержит символ перевода строки \n"  
"C:\\Program Files\\MetaTrader 4"  
"A" "1234567890" "0" "$"
```

Внутреннее представление - структура размером 8 байт. Первый элемент структуры - длинное целое, содержит размер распределенного для строки буфера. Второй элемент структуры - 32-разрядный адрес буфера, содержащего строку

### **1.2.7 Цветовые константы (color)**

Цветовые константы могут быть представлены тремя различными способами: литерально, целочисленно или при помощи имени (только для именованных Web-цветов).

Литеральное представление состоит из трех частей, представляющих числовые значения интенсивности трех основных компонент цвета: красной (red), зеленой (green), синей (blue). Константа начинается с символа C и обрамляется одинарными кавычками.

Числовые значения интенсивности компоненты цвета лежат в диапазоне от 0 до 255.

Целочисленное представление записывается в виде шестнадцатеричного или десятичного числа. Шестнадцатеричное число имеет вид 0x00BBGGRR, где RR - значение интенсивности красной компоненты цвета, GG - зеленой, а BB - синей. Десятичные константы не имеют прямого отражения в RGB. Они представляют собой десятичное значение шестнадцатеричного целочисленного представления.

Именованные цвета отражают так называемый набор Web-цветов.

**Примеры:**

```
// литералы  
C'128,128,128' // серый
```

```

C'0x00,0x00,0xFF' // синий
//названия цветов
Red           // красный
Yellow        // желтый
Black         // черный
// целочисленные представления
0xFFFFFFFF   // белый
16777215     // белый
0x008000     // зеленый
32768        // зеленый

```

Внутреннее представление - длинное целое число размером 4 байта. Первый байт не учитывается. Остальные 3 байта содержат RGB-составляющие.

### **1.2.8 Константы даты и времени (datetime)**

Константы даты и времени могут быть представлены в виде литеральной строки, которая состоит из 6 частей, представляющих числовое значение года, месяца, числа (либо числа, месяца, года), часа, минуты и секунды. Константа обрамляется одинарными кавычками и начинается с символа D. Может опускаться либо дата (год, месяц, число), либо время (часы, минуты, секунды), либо все вместе. Диапазон значений от 1 января 1970 года до 31 декабря 2037 года.

#### **Примеры:**

```

D'2004.01.01 00:00' // Новый Год
D'1980.07.19 12:30:27'
D'19.07.1980 12:30:27'
D'19.07.1980 12' //равнозначно D'1980.07.19 12:00:00'
D'01.01.2004' //равнозначно D'01.01.2004 00:00:00'
D'12:30:27' //равнозначно D'[дата компиляции]
12:30:27'
D'' //равнозначно D'[дата компиляции]
00:00:00'

```

Внутреннее представление - длинное целое число размером 4 байта. Значение представляет собой количество секунд, прошедшее с 00:00 1 января 1970 года.

### **1.3 Операции и выражения**

Некоторым символам и символьным последовательностям придается особое значение. Это - так называемые символы операций, например:

```

+ - * / %           символы арифметических операций

```

&&	символы логических операций
= += *=	символы операций присваивания

Символы операций используются в выражениях и имеют смысл тогда, когда им даны соответствующие операнды

Также особое значение придается знакам препинания. Знаки препинания включают круглые скобки, фигурные скобки, запятую, двоеточие и точку с запятой.

Символы операций, знаки препинания и пробелы служат для того, чтобы отделять элементы языка.

### 1.3.1 Выражения

Выражение состоит из одного или нескольких операндов и символов операций. Может записываться в несколько строк.

#### Примеры:

```
a++; b = 10;
x = (y * z) /
    (w + 2) + 127;
```

Выражение, заканчивающееся точкой с запятой (;), является оператором.

### 1.3.2 Арифметические операции

К арифметическим относятся аддитивные и мультипликативные операции:

Сумма величин	<code>i = j + 2;</code>
Вычитание величин	<code>i = j - 3;</code>
Изменение знака	<code>x = - x;</code>
Умножение величин	<code>z = 3 * x;</code>
Частное от деления	<code>i = j / 5;</code>
Остаток от деления	<code>minutes = time % 60;</code>
Добавление 1 к значению переменной	<code>i++;</code>
Вычитание 1 от значения переменной	<code>k--;</code>

Операции увеличения/уменьшения значения переменной не могут применяться в выражениях.

### Примеры:

```
int a=3;
a++;           // верное выражение
int b=(a++)*3; // неверное выражение
```

### 1.3.3 Операция присваивания

Значением выражения, в которое входит операция присваивания, является значение левого операнда после присваивания:

Присваивание значения x переменной y	y = x;
--------------------------------------	--------

Следующие операции объединяют арифметические или побитовые операции с операцией присваивания:

Увеличение значения переменной y на x	y += x;
Уменьшение значения переменной y на x	y -= x;
Умножение значения переменной y на x	y *= x;
Деление значения переменной y на x	y /= x;
Остаток от деления значения переменной y на x	y %= x;
Сдвиг двоичного представления y вправо на x бит	y >>= x;
Сдвиг двоичного представления y влево на x бит	y <<= x;
Побитовая операция И двоичных представлений y и x	y &= x;
Побитовая операция ИЛИ двоичных представлений y и x	y  = x;
Побитовая операция исключающее ИЛИ двоичных представлений y и x	y ^= x;

В выражении может быть только одна операция присваивания. Побитовые операции производятся только с целыми числами. При выполнении операции логический сдвиг представления y вправо/влево на x бит используются младшие 5 двоичных разрядов

значения x, старшие разряды отбрасываются, то есть сдвиг производится на 0-31 бит. При выполнении операции %= (значение y по модулю x) знак результата совпадает со знаком делимого.

### 1.3.4 Операции отношения

Логическое значение ЛОЖЬ представляется целым нулевым значением, а значение ИСТИНА представляется любым ненулевым.

Значением выражений, содержащих операции отношения или логические операции, являются ЛОЖЬ(0) или ИСТИНА(1).

Истина, если a равно b	a == b;
Истина, если a не равно b	a != b;
Истина, если a меньше b	a < b;
Истина, если a больше b	a > b;
Истина, если a меньше или равно b	a <= b;
Истина, если a больше или равно b	a >= b;

Два ненормализованных числа с плавающей точкой нельзя связывать операциями == или !=. Следует из одного числа вычесть другое и нормализованный результат сравнить с нулем

### 1.3.5 Логические операции

Операнд операции логического отрицания НЕ(!) должен иметь арифметический тип.

Результат равен ИСТИНА(1), если значение операнда есть ЛОЖЬ(0), и равен ЛОЖЬ(0), если операнд не равен ЛОЖЬ(0).

```
if(!a) Print("не 'a'");
```

Логическая операция ИЛИ (||) значений x и y. Значением выражения является ИСТИНА(1), если истинно (не нуль) значение x или y. В противном случае - ЛОЖЬ(0).

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

Логическая операция И (&&) значений x и y. Значением выражения является ИСТИНА(1), если значения x и y истинны (не нуль). В противном случае - ЛОЖЬ(0). Логические выражения вычисляются полностью, т.е., к ним не применяется схема так называемой "короткой оценки".

```
if (p!=x && p>y) Print ("TRUE");
```

### 1.3.6 Побитовые операции

Дополнение до единицы значения переменной. Значение выражения содержит 1 во всех разрядах, в которых значение переменной содержит 0, и 0 во всех разрядах, в которых значения переменной содержит 1.

```
b = ~n;
```

Двоичное представление  $x$  сдвигается вправо на  $y$  разрядов. Сдвиг вправо логический, то есть освобождающиеся слева разряды будут заполняться нулями.

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

Двоичное представление  $x$  сдвигается влево на  $y$  разрядов; освобождающиеся справа разряды заполняются нулями.

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

Побитовая операция И двоичных представлений  $x$  и  $y$ . Значение выражения содержит 1 (ИСТИНА) во всех разрядах, в которых  $x$  и  $y$  содержат не ноль; и 0 (ЛОЖЬ) во всех остальных разрядах.

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

Побитовая операция ИЛИ двоичных представлений  $x$  и  $y$ . Значение выражения содержит 1 во всех разрядах, в которых  $x$  или  $y$  не содержит 0, и 0 - во всех остальных разрядах.

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

Побитовая операция исключающее ИЛИ (eXclusive OR) двоичных представлений  $x$  и  $y$ . Значение выражения содержит 1 в тех разрядах, в которых  $x$  и  $y$  имеют разные двоичные значения, и 0 - во всех остальных разрядах.

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

Побитовые операции выполняются только с целыми числами

### 1.3.7 Другие операции

#### Индексирование

При обращении к *i*-му элементу массива значением выражения является значение переменной с порядковым номером *i*.

#### Пример:

```
array[i] = 3; // Присвоить значение 3 i-му элементу массива array.
```

Индексом массива может быть только целое число. Допускаются не более чем четырехмерные массивы. Индексация каждого измерения производится от 0 до **размер измерения-1**. В частном случае одномерного массива из 50 элементов обращение к первому элементу будет выглядеть как `array[0]`, к последнему элементу - `array[49]`.

При доступе за пределы массива исполняющая подсистема сгенерирует ошибку `ERR_ARRAY_INDEX_OUT_OF_RANGE` (4002), которую можно получить при помощи функции `GetLastError()`.

#### Вызов функции с аргументами *x1, x2, ..., xn*

Каждый аргумент может представлять собой константу, переменную или выражение соответствующего типа. Передаваемые аргументы разделяются запятыми и должны находиться внутри круглых скобок, открывающая круглая скобка должна следовать за именем вызываемой функции.

Значением выражения является значение, возвращаемое функцией. Если тип возвращаемого значения функции есть `void`, то вызов такой функции нельзя помещать справа в операции присвоения. Обратите внимание, что порядок выполнения выражений *x1, ..., xn* гарантируется.

#### Пример:

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

### Операция запятая

Выражения, разделенные запятыми, вычисляются слева направо. Все побочные эффекты вычисления левого выражения могут возникать до вычисления правого выражения. Тип и значение результата совпадают с типом и значением правого выражения. В качестве примера можно рассматривать список передаваемых параметров (см.выше).

#### Пример:

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

### 1.3.8 Приоритеты и порядок выполнения операций

Для каждой группы операций в таблице приоритет одинаков. Чем выше приоритет группы операций, тем выше она расположена в таблице. Порядок выполнения определяет группировку операций и операндов.

()	Вызов функции	Слева направо
[]	Выделение элемента массива	
!	Логическое отрицание	Справа налево
-	Изменение знака	
++	Увеличение на единицу (increment)	
--	Уменьшение на единицу (decrement)	
~	Побитовое отрицание (complement)	
&	Побитовая операция И	Слева направо
	Побитовая операция ИЛИ	
^	Побитовая операция исключающее ИЛИ (eXclude OR)	
<<	Сдвиг влево	
>>	Сдвиг вправо	
*	Умножение	Слева направо
/	Деление	
%	Деление по модулю	
+	Сложение	Слева направо
-	Вычитание	
<	Меньше, чем	Слева направо
<=	Меньше или равно	
>	Больше, чем	

>=	Больше или равно	
==	Равно	
!=	Не равно	
	Логическая операция ИЛИ	Слева направо
&&	Логическая операция И	Слева направо
=	Присваивание	Справа налево
+=	Сложение с присваиванием	
-=	Вычитание с присваиванием	
*=	Умножение с присваиванием	
/=	Деление с присваиванием	
%=	Деление по модулю с присваиванием	
>>=	Сдвиг вправо с присваиванием	
<<=	Сдвиг влево с присваиванием	
&=	Побитовое И с присваиванием	
=	Побитовое ИЛИ с присваиванием	
^=	Исключающее ИЛИ с присваиванием	
,	Запятая	Слева направо

Для изменения порядка выполнения операций применяются круглые скобки, которые имеют высший приоритет.

**Внимание:** приоритет выполнения операций в языке MQL4 несколько отличается от приоритета, принятого в языке Си

## **1.4 Операторы**

Операторы языка описывают некоторые алгоритмические действия, которые необходимо выполнить для решения задачи. Тело программы - это последовательность таких операторов. Идущие друг за другом операторы разделяются точкой с запятой.

Один оператор может занимать одну или более строк. Два или большее количество операторов могут быть расположены на одной строке. Операторы, управляющие порядком выполнения (if, if-else, switch, while и for), могут быть вложены друг в друга.

**Пример:**

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

### 1.4.1 Составной оператор

Составной оператор (блок) состоит из одного или большего числа операторов любого типа, заключенных в фигурные скобки { }. После закрывающейся фигурной скобки не должно быть точки с запятой (;).

#### Пример:

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

### 1.4.2 Оператор-выражение

Любое выражение, заканчивающееся точкой с запятой (;), является оператором. Далее следуют примеры операторов-выражений.

#### Оператор присваивания:

```
Идентификатор = выражение;

x=3;
y=x=3; // ошибка
```

В выражении оператор присваивания может использоваться только один раз.

#### Оператор вызова функции:

```
Имя_функции (аргумент1, ..., аргументN);

FileClose(file);
```

## Пустой оператор

Состоит только из точки с запятой (;) и используется для обозначения пустого тела управляющего оператора

### 1.4.3 Оператор завершения **break**

Оператор *break* прекращает выполнение ближайшего вложенного внешнего оператора *switch*, *while* или *for*. Управление передается оператору, следующему за заканчиваемым. Одно из назначений этого оператора - закончить выполнение цикла при присваивании некоторой переменной определенного значения.

#### Пример:

```
// поиск первого нулевого элемента
for(i=0;i<array_size;i++)
    if(array[i]==0)
        break;
```

### 1.4.4 Оператор продолжения **continue**

Оператор *continue* передает управление в начало ближайшего внешнего оператора цикла *while* или *for*, вызывая начало следующей итерации. Этот оператор по действию противоположен оператору *break*.

#### Пример:

```
// сумма всех ненулевых элементов
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);
}
```

```
}
```

### **1.4.5 Оператор возврата return**

Оператор *return* прекращает выполнение текущей функции и возвращает управление вызвавшей программе. Использование *return(выражение)*; прекращает выполнение текущей функции с передачей результата. Выражение оператора заключается в круглые скобки и не должно содержать оператор присваивания.

#### **Пример:**

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

В функциях с типом возвращаемого значения `void` необходимо использовать оператор *return* без выражения:

```
void SomeFunction()
{
    Print("Hello!");
    return;    // этот оператор можно удалить
}
```

Завершающая фигурная скобка функции предполагает неявное исполнение оператора *return* без выражения

### **1.4.6 Условный оператор if-else**

Если выражение истинно, то выполняется оператор1 и управление передается на оператор, следующий за оператором2 (т. е. оператор2 не выполняется). Если выражение ложно, то выполняется оператор2.

```
if (выражение)
    оператор1
else
```

Часть *else* оператора *if* может опускаться. Поэтому во вложенных операторах *if* с пропущенной частью *else* может возникнуть неоднозначность. В этом случае *else* связывается с ближайшим предыдущим оператором *if* в том же блоке, не имеющим части *else*.

**Примеры:**

```
// Часть else относится ко второму оператору if:
if(x>1)
    if(y==2) z=5;
    else     z=6;

// Часть else относится к первому оператору if
if(x>1)
{
    if(y==2) z=5;
}
else     z=6;

// Вложенные операторы
if(x=='a')
{
    y=1;
}
else if(x=='b')
{
    y=2;
    z=3;
}
else if(x=='c')
```

```
{
    y = 4;
}
else Print("ERROR");
```

### 1.4.7 Оператор-переключатель switch

Сравнивает значение выражения с константами во всех вариантах *case* и передает управление оператору, который соответствует значению выражения. Каждый вариант *case* может быть помечен целой константой, символьной константой или константным выражением. Константное выражение не может включать переменные или вызовы функций. Выражение оператора *switch* должно быть целого типа.

```
switch (выражение)
{
    case константа: операторы
    case константа: операторы
    ...
    default: операторы
}
```

Операторы, связанные с меткой *default*, выполняются, если ни одна из констант в операторах *case* не равна значению выражения. Вариант *default* обязательно должен быть последним. Если ни одна константа не соответствует значению выражения и вариант *default* отсутствует, то не выполняется никаких действий. Ключевое слово *case* вместе с константой служат просто метками, и если будут выполняться операторы для некоторого варианта *case*, то далее будут выполняться операторы всех последующих вариантов до тех пор, пока не встретится оператор *break*, что позволяет связывать одну последовательность операторов с несколькими вариантами.

Константное выражение вычисляется в период компиляции. Никакие две константы в одном операторе-переключателе не могут иметь одинаковые значения.

#### **Пример:**

```
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;
}

```

#### **1.4.8 Оператор цикла while**

Если выражение истинно, то оператор выполняется до тех пор, пока выражение не станет ложным. Если выражение ложно, то управление передается следующему оператору.

```

while (выражение)
    оператор;

```

Значение выражения определяется до выполнения оператора. Следовательно, если выражение ложно с самого начала, то оператор вообще не выполняется.

#### **Пример:**

```

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

```

#### **1.4.9 Оператор цикла for**

*Выражение1* описывает инициализацию цикла. *Выражение2* - проверка условия завершения цикла. Если оно истинно, то выполняется оператор тела цикла *for*. Все

повторяется, пока *выражение2* не станет ложным. Если оно ложно, цикл заканчивается и управление передается следующему оператору. *Выражение3* вычисляется после каждой итерации.

```
for (выражение1; выражение2; выражение3)
    оператор;
```

Оператор *for* эквивалентен следующей последовательности операторов:

```
выражение1;
while (выражение2)
{
    оператор;
    выражение 3;
};
```

Любое из трех или все три выражения в операторе *for* могут отсутствовать, однако разделяющие их точки с запятыми (;) опускать нельзя. Если опущено *выражение2*, то считается, что оно постоянно истинно. Оператор *for(;;)* представляет собой бесконечный цикл, эквивалентный оператору *while(1)*. Каждое из *выражение1* и *выражение3* может состоять из нескольких выражений, объединенных оператором запятая ','.

### Примеры:

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

for (;;)
{
    Print (MathPower (x, 2) );
    x++;
    if (x>10) break;
}

for (i=0, j=n-1; i<n; i++, j--) a[i]=a[j];
```

## 1.5 Функции

Функция - это поименованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо. Она состоит из описания типа возвращаемого значения, имени, формальных параметров и составного оператора (блока) из выполняемых действий. Количество параметров, передаваемых в функцию, ограничено и не может превышать 64.

### Пример:

```
double                // тип возвращаемого значения
lfunc (double a, double b) // имя функции и список
параметров
{
    // составной оператор
    return (a + b);      // возвращаемое значение
}
```

Оператор `return` может возвращать значение выражения, стоящего в этом операторе. Значение выражения при необходимости преобразуется к типу результата функции. Функция, которая не возвращает значения, должна быть описана как имеющая тип `void`.

### Пример:

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

Параметры, передаваемые в функцию могут иметь умолчательные значения, которые задаются константами соответствующего типа.

### Пример:

```
int somefunc(double a, double d=0.0001, int n=5, bool b=true,
string s="passed string")
{
    Print("Обязательный параметр a=", a);
    Print("Переданы следующие параметры: d=", d, " n=", n, "
b=", b, " s=", s);
    return (0);
}
```

```
}
```

Если какому-либо параметру было назначено умолчательное значение, то все последующие параметры также должны иметь умолчательное значение.

### Пример неправильного объявления:

```
int somefunc(double a, double d=0.0001, int n, bool b, string
s="passed string")
{
}
```

## 1.5.1 Вызов функции

Если некоторое имя, которое не было описано ранее, появляется в выражении и за ним следует левая круглая скобка, то оно по контексту считается именем некоторой функции.

```
имя_функции (x1, x2, ..., xn)
```

Аргументы (формальные параметры) передаются по значению, т. е. каждое выражение  $x_1, \dots, x_n$  вычисляется и значение передается функции. Порядок вычисления выражений и порядок загрузки значений гарантируются. Во время выполнения производится проверка числа и типа аргументов, переданных функции. Такой способ обращения к функции называется вызовом по значению. Вызов функции - это выражение, значением которого является значение, возвращаемое функцией. Описанный тип функции должен соответствовать типу возвращаемого значения. Функция может быть объявлена или описана в любом месте программы на глобальном уровне, то есть, вне других функций. Функция не может быть объявлена или описана внутри другой функции.

### Примеры:

```
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);
}
```

При вызове функции, имеющей умолчательные параметры, список передаваемых параметров можно ограничить не ранее первого умолчательного параметра.

**Примеры:**

```
void somefunc(double init,double sec=0.0001,int level=10); // прототип
функции

somefunc(); // неправильный вызов. первый обязательный
параметр должен быть.

somefunc(3.14); // правильный вызов
somefunc(3.14, 0.0002); // правильный вызов
somefunc(3.14, 0.0002, 10); // правильный вызов
```

При вызове функции, нельзя пропускать параметры, даже имеющие умолчательные значения:

```
somefunc(3.14, , 10); // неправильный вызов. второй параметр
пропущен
```

### **1.5.2 Специальные функции**

В MQL4 существуют 3 функции с predefined именами:

**init()** - функция, вызываемая в процессе инициализации модуля. В случае ее отсутствия при инициализации не вызывается никакой функции.

**start()** - основная функция. У экспертов вызывается после прихода очередного тика. У пользовательских индикаторов вызывается при пересчете после прикрепления индикатора к графику, при открытии клиентского терминала (если индикатор прикреплен к графику), а также после прихода очередного тика. У скриптов выполняется сразу после прикрепления к графику и выполнения инициализации. В случае отсутствия в модуле функции start() этот модуль (эксперт, скрипт или пользовательский индикатор) не может быть запущен.

**deinit()** - функция, вызываемая в процессе деинициализации модуля. В случае ее отсутствия при деинициализации не вызывается никакой функции.

Предопределенные функции могут иметь параметры. Однако при вызове этих функций клиентским терминалом никакие параметры переданы извне не будут, а будут использованы умолчательные значения. Функции *start()*, *init()* и *deinit()* могут быть вызваны из любого места модуля по общим правилам, наравне с другими функциями.

Нежелательно из функции *init()* вызывать *start()* или совершать торговые операции, так как в момент инициализации модуля могут быть не готовы данные графиков, рыночные цены и т.д. Функции *init()* и *deinit()* должны максимально быстро завершать свою работу и ни в коем случае не зацикливаться в попытке начать полноценную работу раньше вызова функции *start()*

## **1.6 Переменные**

Переменные должны быть объявлены перед их использованием. Для идентификации переменных используются уникальные имена. Описания переменных используются для их определения и объявления типов. Описание не является оператором.

**Основными типами являются:**

- int - целые числа;
- bool - логические значения *true* и *false*;
- string - символьные строки;
- double - числа двойной точности с плавающей точкой.

**Примеры:**

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

**Дополнительные типы:**

- color - целое число, представляющее RGB-цвет;
- datetime - дата и время, беззнаковое целое число, содержащее количество секунд, прошедших с 0 часов 1 января 1970 года.

Дополнительные типы данных имеют смысл только при объявлении входных параметров для более удобного представления их в окне свойств.

**Примеры:**

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

**Массивы**

Массив - это индексированная совокупность однотипных данных:

```
int     a[50];           // Одномерный массив из 50 целых чисел.  
double m[7][50];        // Двухмерный массив из семи массивов,  
                        // каждый из которых состоит из 50 чисел.
```

Индексом массива может быть только целое число. Допускаются не более чем четырехмерные массивы. Нумерация элементов массива начинается с 0. Последний элемент одномерного массива имеет номер на 1 меньший, чем размер массива, то есть обращение к последнему элементу массива из 50 целых чисел будет выглядеть как a[49]. То же самое относится и к многомерным массивам - индексация одного измерения производится от 0 до **размер измерения-1**. Последний элемент двумерного массива из примера будет выглядеть как m[6][49].

При доступе за пределы массива исполняющая подсистема сгенерирует ошибку ERR\_ARRAY\_INDEX\_OUT\_OF\_RANGE (4002), которую можно получить при помощи функции GetLastError().

### **1.6.1 Локальные переменные**

Переменная, объявленная внутри какой-либо функции, является локальной. Область видимости локальной переменной ограничена пределами функции, внутри которой она объявлена. Локальная переменная может быть проинициализирована при помощи любого выражения. Инициализация локальной переменной производится каждый раз при вызове соответствующей функции. Локальные переменные располагаются во временной области памяти соответствующей функции.

**Пример:**

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

## 1.6.2 Формальные параметры

Передаваемые в функцию параметры являются локальными. Областью видимости является блок функции. Формальные параметры должны отличаться по именам от внешних переменных и локальных переменных, определенных внутри функции. В блоке функции формальным параметрам могут быть присвоены некоторые значения.

**Пример:**

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

Формальные параметры могут быть проинициализированы константами. В этом случае инициализирующее значение считается значением по умолчанию. Параметры, следующие за проинициализированным параметром, должны быть тоже проинициализированы.

**Пример:**

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

При вызове такой функции инициализированные параметры можно опускать, вместо них будут подставлены значения по умолчанию.

**Пример:**

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

Библиотечные функции, импортируемые в других модулях, не могут иметь параметров по умолчанию.

Параметры передаются по значению, то есть изменения соответствующей локальной переменной внутри вызываемой функции никак не отразится в вызывающей функции. В качестве параметров можно передавать массивы. Но у массива, переданного в качестве параметра, нельзя изменять значения его элементов.

Существует возможность передавать параметры по ссылке. В этом случае модификация таких параметров отразится на соответствующих переменных в вызываемой функции, переданных по ссылке. Нельзя передавать по ссылке элементы массивов. Параметры по ссылке можно передавать только в пределах одного модуля, для библиотечных функций такая возможность не предусмотрена. Для того чтобы указать, что параметр передается по ссылке, после типа данных необходимо поставить модификатор `&`.

**Пример:**

```
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;
}
```

Массивы также можно передавать по ссылке, все изменения отразятся в исходном массиве. В отличие от простых параметров массивы можно передавать по ссылке и в библиотечные функции.

Параметры, передаваемые по ссылке, нельзя инициализировать значениями по умолчанию.

В функцию нельзя передать больше 64 параметров

### 1.6.3 Статические переменные

Передаваемые в функцию параметры являются локальными. Областью видимости является блок функции. Формальные параметры должны отличаться по именам от внешних переменных и локальных переменных, определенных внутри функции. В блоке функции формальным параметрам могут быть присвоены некоторые значения.

**Пример:**

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

Формальные параметры могут быть проинициализированы константами. В этом случае инициализирующее значение считается значением по умолчанию. Параметры, следующие за проинициализированным параметром, должны быть тоже проинициализированы.

**Пример:**

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

При вызове такой функции инициализированные параметры можно опускать, вместо них будут подставлены значения по умолчанию.

**Пример:**

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

Библиотечные функции, импортируемые в других модулях, не могут иметь параметров по умолчанию.

Параметры передаются по значению, то есть изменения соответствующей локальной переменной внутри вызываемой функции никак не отразится в вызывающей функции. В качестве параметров можно передавать массивы. Но у массива, переданного в качестве параметра, нельзя изменять значения его элементов.

Существует возможность передавать параметры по ссылке. В этом случае модификация таких параметров отразится на соответствующих переменных в вызываемой функции,

переданных по ссылке. Нельзя передавать по ссылке элементы массивов. Параметры по ссылке можно передавать только в пределах одного модуля, для библиотечных функций такая возможность не предусмотрена. Для того чтобы указать, что параметр передается по ссылке, после типа данных необходимо поставить модификатор &.

**Пример:**

```
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;
}
```

Массивы также можно передавать по ссылке, все изменения отразятся в исходном массиве. В отличие от простых параметров массивы можно передавать по ссылке и в библиотечные функции.

Параметры, передаваемые по ссылке, нельзя инициализировать значениями по умолчанию.

В функцию нельзя передать больше 64 параметров

#### **1.6.4 Глобальные переменные**

Глобальные переменные определяются на том же уровне, что и функции, т. е. не локальны ни в каком блоке.

**Пример:**

```
int GlobalFlag=10;    // глобальная переменная

int start()
{
```

```
...  
}
```

Область видимости глобальных переменных - вся программа, глобальные переменные доступны из всех функций, определенных в программе. Инициализируются нулем, если явно не задано другое начальное значение. Глобальная переменная может быть проинициализирована только соответствующей ее типу константой. Инициализация глобальных переменных производится однократно сразу после загрузки программы в память клиентского терминала.

Замечание: не следует путать переменные, объявленные на глобальном уровне, с глобальными переменными клиентского терминала, доступ к которым осуществляется при помощи функций *GlobalVariable...()*.

### **1.6.5 Внешние переменные**

Глобальные переменные определяются на том же уровне, что и функции, т. е. не локальны ни в каком блоке.

#### **Пример:**

```
int GlobalFlag=10;    // глобальная переменная  
  
int start()  
{  
    ...  
}
```

Область видимости глобальных переменных - вся программа, глобальные переменные доступны из всех функций, определенных в программе. Инициализируются нулем, если явно не задано другое начальное значение. Глобальная переменная может быть проинициализирована только соответствующей ее типу константой. Инициализация глобальных переменных производится однократно сразу после загрузки программы в память клиентского терминала.

Замечание: не следует путать переменные, объявленные на глобальном уровне, с глобальными переменными клиентского терминала, доступ к которым осуществляется при помощи функций *GlobalVariable...()*.

### **1.6.6 Инициализация переменных**

Любая переменная при определении может быть инициализирована. Любая переменная инициализируется нулем (0), если явно не задано другое начальное значение. Глобальные и статические переменные могут быть проинициализированы только константой соответствующего типа. Локальные переменные могут быть проинициализированы любым выражением, а не только константой.

Инициализация глобальных и статических переменных производится однократно. Инициализация локальных переменных производится каждый раз при вызове соответствующих функций.

#### **Примеры:**

```
int    n        = 1;
double p        = MarketInfo(Symbol(), MODE_POINT);
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 };

```

Список значений элементов массива должен быть заключен в фигурные скобки. Пропущенные инициализирующие значения считаются равными 0. Если размер инициализируемого массива не указан, то он определяется компилятором, исходя из размера инициализирующей последовательности. Многомерные массивы инициализируются одномерной последовательностью, последовательностью без дополнительных фигурных скобок. Массивы (в том числе и объявленные на локальном уровне) могут инициализироваться только константами

### **1.6.7 Описание внешних функций**

Тип внешних функций, определенных в другом модуле, должен быть явно описан.

Отсутствие такого описания может привести к ошибкам при компиляции, компоновке или

выполнении программы. При описании внешнего объекта используйте ключевое слово *#import* с указанием модуля.

### Примеры:

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

С помощью импорта можно очень легко описывать функции, вызываемые из внешних DLL или скомпилированных EX4 библиотек.

Существует способ передавать в импортируемые dll-функции указатели на переменные. Данные типа *string* передаются как указатель на соответствующую область памяти (напомним, что внутреннее представление строковых данных состоит из двух частей: длины области памяти и указателя на область памяти). Если необходимо передать данные типа *int* или *double*, то в качестве параметра следует передать по ссылке одноэлементный массив соответствующего типа.

### Пример:

```
#import "some_lib.dll"
    void    PassIntegerByref(int& OneInt[]);
#import
int start()
{
    int array[1];
//...
    PassIntegerByref(array);
    Print(array[0]);
//...
```

```
}
```

## **1.7 Препроцессор**

Препроцессор - это специальная подсистема компилятора MQL4, которая занимается предварительной подготовкой исходного текста программы непосредственно перед ее компиляцией.

Препроцессор позволяет улучшить читаемость исходного кода. Структурирование кода может быть достигнуто путем включения отдельных файлов с исходными кодами MQL4-программ. Улучшению читаемости кода способствует и возможность присвоения мнемонических имен отдельным константам.

Препроцессор позволяет также определять специфические параметры MQL4-программ.

Если в качестве первого символа в строке программы используется символ #, то эта строка является директивой препроцессора. Директива препроцессора заканчивается символом перевода на новую строку

### **1.7.1 Объявление константы**

Используя конструкцию *#define*, можно в начале программы определить символическое имя или символическую константу, которая будет конкретной строкой символов.

Впоследствии компилятор заменит все не заключенные в кавычки появления этого имени на соответствующую строку. Фактически это имя может быть заменено абсолютно произвольным текстом, не обязательно цифрами:

```
#define идентификатор значение
```

Идентификатор константы подчиняется тем же правилам, что и для имен переменных.

Значение может быть любого типа:

```
#define ABC          100
#define PI           0.314
#define COMPANY_NAME "MetaQuotes Software Corp."
```

```

...

void ShowCopyright()
{
    Print("Copyright ? 2001-2007, ",COMPANY_NAME);
    Print("http://www.metaquotes.net");
}

```

### 1.7.2 Специальные параметры

У каждой MQL4 программы можно указать дополнительные специфические параметры *#property*, которые помогают клиентскому терминалу правильно обслуживать программы без необходимости их явного запуска. В первую очередь это касается внешних настроек индикаторов.

*#property* идентификатор значение

Константа	Тип	Описание
link	string	ссылка на сайт компании-производителя
copyright	string	название компании-производителя
stacksize	int	размер стека для рекурсивных вызовов
library		библиотека; не назначается никакой стартовой функции, не удаляются функции, которые не вызываются из других функций
indicator_chart_window	void	выводить индикатор в окно графика
indicator_separate_window	void	выводить индикатор в отдельное окно
indicator_buffers	int	количество буферов для расчета индикатора, максимум до 8
indicator_minimum	double	нижнее ограничение шкалы отдельного окна индикатора

indicator_maximum	double	верхнее ограничение шкалы отдельного окна индикатора
indicator_colorN	color	цвет для вывода линии N, где N от 1 до 8
indicator_widthN	int	толщина линии N, где N от 1 до 8
indicator_styleN	int	стиль линии N, где N от 1 до 8
indicator_levelN	double	горизонтальный уровень N в отдельном окне индикатора, где N от 1 до 8
indicator_levelcolor	color	цвет горизонтальных уровней индикатора
indicator_levelwidth	int	толщина горизонтальных уровней индикатора
indicator_levelstyle	int	стиль горизонтальных уровней индикатора
show_confirm	void	выводить окно подтверждения перед запуском скрипта
show_inputs	void	выводить окно со свойствами перед запуском скрипта и запретить вывод окна подтверждения

### Примеры:

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

Компилятор запишет в настройках выполняемого модуля объявленные значения.

### 1.7.3 Включение файлов

Командная строка *#include* может встречаться в любом месте программы, но обычно все включения размещаются в начале файла исходного текста. Формат вызова:

```
#include <имя_файла>
```

```
#include "имя_файла";
```

### Примеры:

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

Препроцессор заменяет строку `#include <имя_файла>` содержимым файла `WinUser32.mqh`. Угловые скобки обозначают, что файл `WinUser32.mqh` будет взят из стандартного каталога (обычно это `каталог_терминала\experts\include`). Текущий каталог не просматривается.

Если имя файла заключено в кавычки, то поиск производится в текущем каталоге (в котором содержится основной файл исходного текста). Стандартный каталог не просматривается

#### 1.7.4 Импорт функций

Импорт функций осуществляется из откомпилированных модулей MQL4 (файлы \*.ex4) и из модулей операционной системы (файлы \*.dll). Имя модуля указывается в директиве `#import`. Для того, чтобы компилятор мог правильно оформить вызов импортируемой функции и организовать правильную передачу параметров, необходимо полное описание функций. Описания функций следуют непосредственно за директивой `#import "имя модуля"`. Новая команда `#import` (можно без параметров) завершает блок описания импортируемых функций.

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

Импортируемые функции должны иметь уникальные имена. Нельзя одновременно импортировать из разных модулей функции с одинаковыми именами. Импортируемые функции не должны иметь имен, совпадающих с именами встроенных функций.

Так как импортируемые функции находятся вне компилируемого модуля, компилятор не может проверить правильность передаваемых параметров. Поэтому, во избежание ошибок выполнения, необходимо точно описывать состав и порядок параметров, передаваемых в импортируемые функции. Параметры, передаваемые в импортируемые функции (как из EX4, так и из DLL-модулей), не могут иметь значения по умолчанию.

### Примеры:

```
#import "user32.dll"
    int    MessageBoxA(int hWnd, string lpText, string
lpCaption, int uType);

#import "stdlib.ex4"
    string ErrorDescription(int error_code);
    int    RGB(int red_value, int green_value, int blue_value);
    bool   CompareDoubles(double number1, double number2);
    string DoubleToStrMorePrecision(double number, int
precision);
    string IntegerToHexString(int integer_number);

#import "ExpertSample.dll"
    int    GetIntValue(int);
    double GetDoubleValue(double);
    string GetStringValue(string);
    double GetArrayItemValue(double arr[], int, int);
    bool   SetArrayItemValue(double& arr[], int,int, double);
    double GetRatesItemValue(double rates[][6], int, int, int);
    int    SortStringArray(string& arr[], int);
    int    ProcessStringArray(string& arr[], int);

#import
```

Для импорта функций во время выполнения mql4-программы используется так называемое позднее связывание. Это значит, что пока не вызвана импортируемая функция, соответствующий модуль (ex4 или dll) не загружается.

Не рекомендуется использовать полностью квалифицированное имя загружаемого модуля вида *Drive:\Directory\FileName.Ext*. Библиотеки MQL4 загружаются из папки *terminal\_dir\experts\libraries*. Если библиотека не была найдена, то производится попытка загрузить библиотеку из папки *terminal\_dir\experts*

## **2 Стандартные константы**

Для облегчения написания программ, а также для удобства восприятия исходных текстов программ, в языке MQL4 предусмотрены предопределенные стандартные константы.

Стандартные константы являются аналогом макроподстановок и имеют тип `int`.

Константы сгруппированы по своему назначению

### **2.1 Таймсерии**

Идентификаторы таймсерий используются в функциях `ArrayCopySeries()`, `iHighest()` и `iLowest()`.

Могут быть одной из следующих величин:

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
MODE_OPEN	0	Цена открытия
MODE_LOW	1	Минимальная цена
MODE_HIGH	2	Максимальная цена
MODE_CLOSE	3	Цена закрытия
MODE_VOLUME	4	Объем (количество тиков, сформировавших бар)
MODE_TIME	5	Время открытия бара

### **2.2 Периоды графиков**

Период графика может быть любым из следующих величин:

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
PERIOD_M1	1	1 минута
PERIOD_M5	5	5 минут
PERIOD_M15	15	15 минут
PERIOD_M30	30	30 минут
PERIOD_H1	60	1 час

PERIOD_H4	240	4 часа
PERIOD_D1	1440	1 день
PERIOD_W1	10080	1 неделя
PERIOD_MN1	43200	1 месяц
0 (ноль)	0	Период текущего графика

### **2.3 Торговые операции**

Тип операций для функции `OrderSend()`. Может быть любым из следующих величин:

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
OP_BUY	0	Покупка
OP_SELL	1	Продажа
OP_BUYLIMIT	2	Отложенный ордер BUY LIMIT
OP_SELLLIMIT	3	Отложенный ордер SELL LIMIT
OP_BUYSTOP	4	Отложенный ордер BUY STOP
OP_SELLSTOP	5	Отложенный ордер SELL STOP

### **2.4 Ценовые константы**

Используемая цена для расчёта индикаторов может принимать любое из следующих значений:

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
PRICE_CLOSE	0	Цена закрытия
PRICE_OPEN	1	Цена открытия
PRICE_HIGH	2	Максимальная цена
PRICE_LOW	3	Минимальная цена
PRICE_MEDIAN	4	Средняя цена, $(high+low)/2$

PRICE_TYPICAL	5	Типичная цена, $(high+low+close)/3$
PRICE_WEIGHTED	6	Взвешенная цена закрытия, $(high+low+close+close)/4$

## 2.5 MarketInfo

Идентификаторы запроса, используемые в функции MarketInfo(). Могут быть одной из следующего величин:

Константа	Значение	Описание
MODE_LOW	1	Минимальная дневная цена
MODE_HIGH	2	Максимальная дневная цена
MODE_TIME	5	Время поступления последней котировки
MODE_BID	9	Последняя поступившая цена предложения. Для текущего инструмента хранится в предопределенной переменной <u>Bid</u>
MODE_ASK	10	Последняя поступившая цена продажи. Для текущего инструмента хранится в предопределенной переменной <u>Ask</u>
MODE_POINT	11	Размер пункта в валюте котировки. Для текущего инструмента хранится в предопределенной переменной <u>Point</u>
MODE_DIGITS	12	Количество цифр после десятичного точки в цене инструмента. Для текущего инструмента хранится в предопределенной переменной

		<u>Digits</u>
MODE_SPREAD	13	Спрэд в пунктах
MODE_STOPLEVEL	14	Минимально допустимый уровень стоп-лосса/тейк-профита в пунктах
MODE_LOTSIZE	15	Размер контракта в базовой валюте инструмента
MODE_TICKVALUE	16	Размер минимального изменения цены инструмента в валюте депозита
MODE_TICKSIZE	17	Минимальный шаг изменения цены инструмента в пунктах
MODE_SWAPLONG	18	Размер свопа для длинных позиций
MODE_SWAPSHORT	19	Размер свопа для коротких позиций
MODE_STARTING	20	Календарная дата начала торгов (обычно используется для фьючерсов)
MODE_EXPIRATION	21	Календарная дата конца торгов (обычно используется для фьючерсов)
MODE_TRADEALLOWED	22	Разрешение торгов по указанному инструменту
MODE_MINLOT	23	Минимальный размер лота
MODE_LOTSTEP	24	Шаг изменения размера лота
MODE_MAXLOT	25	Максимальный размер лота
MODE_SWAPTYPE	26	Метод вычисления свопов. 0 - в пунктах; 1 - в базовой валюте инструмента; 2 - в процентах; 3 - в валюте залоговых средств.

MODE_PROFITCALCMODE	27	Способ расчета прибыли. 0 - Forex; 1 - CFD; 2 - Futures
MODE_MARGINCALCMODE	28	Способ расчета залоговых средств. 0 - Forex; 1 - CFD; 2 - Futures; 3 - CFD на индексы
MODE_MARGININIT	29	Начальные залоговые требования для 1 лота
MODE_MARGINMAINTENANCE	30	Размер залоговых средств для поддержки открытых позиций в расчете на 1 лот
MODE_MARGINHEDGED	31	Маржа, взимаемая с перекрытых позиций в расчете на 1 лот
MODE_MARGINREQUIRED	32	Размер свободных средств, необходимых для открытия 1 лота на покупку
MODE_FREEZELEVEL	33	Уровень заморозки ордеров в пунктах. Если цена исполнения находится в пределах, определяемых уровнем заморозки, то ордер не может быть модифицирован, отменен или закрыт

## **2.6 Стили рисования**

Перечисление стилей рисования для функций [SetIndexStyle\(\)](#) и [SetLevelStyle\(\)](#). Может быть любым из следующего величин:

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
DRAW_LINE	0	Простая линия

DRAW_SECTION	1	Отрезки между непустыми значениями линии
DRAW_HISTOGRAM	2	Гистограмма
DRAW_ARROW	3	Стрелки (символы)
DRAW_ZIGZAG	4	Отрезки между непустыми значениями чётной и нечётной линий (зигзаг)
DRAW_NONE	12	Отсутствие какого-либо рисования

Стиль линии. Используется только при толщине линии 0 или 1. Может быть любое из следующих значений:

Константа	Значение	Описание
STYLE_SOLID	0	Сплошная линия
STYLE_DASH	1	Штриховая линия
STYLE_DOT	2	Пунктирная линия
STYLE_DASHDOT	3	Штрих-пунктирная линия
STYLE_DASHDOTDOT	4	Штрих-пунктирная линия с двойными точками

Стиль линии может также использоваться для получения или установки свойства OBJPROP\_STYLE объекта

## **2.7 Коды стрелок**

Константы кодов стрелок. Могут иметь следующие значения:

Константа	Значение	Описание
SYMBOL_THUMBSUP	67	Символ - большой палец вверх (👍)
SYMBOL_THUMBSDOWN	68	Символ - большой палец вниз (👎)

SYMBOL_ARROWUP	241	Символ - стрелка вверх (↑)
SYMBOL_ARROWDOWN	242	Символ - стрелка вниз (↓)
SYMBOL_STOPSIGN	251	Символ - СТОП (✖)
SYMBOL_CHECKSIGN	252	Символ - галочка (✓)

Специальные коды стрелок, которые точно указывают на цену и время. Могут быть следующими величинами:

Константа	Значение	Описание
	1	Восходящая стрелка с подсказкой направо (↗)
	2	Нисходящая стрелка с подсказкой направо (↘)
	3	Левый указывающий треугольник (◀)
	4	Символ Черточки (–)
SYMBOL_LEFTPRICE	5	Левая ценовая метка
SYMBOL_RIGHTPRICE	6	Правая ценовая метка

Специальные коды стрелок не могут использоваться в пользовательских индикаторах при установке значения стрелки для линий, имеющих стиль DRAW\_ARROW

## 2.8 Wingdings

Символы шрифта Wingdings, используемые с объектом OBJ\_ARROW:

	3		3		3		3		3		3		3		3		4		4		4		4		4		4		4		4	
	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7																
	4		4		5		5		5		5		5		5		5		5		5		5		5		6		6		6	

	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3
✋	6 4	✋ 5	✋ 6	✋ 7	✋ 8	✋ 9	✋ 0	✋ 1	✋ 2	✋ 3	☺ 4	☹ 5	☹ 6	☹ 7	☹ 8	☹ 9
✋	8 0	✋ 1	☀ 2	☹ 3	☀ 4	✋ 5	✋ 6	✋ 7	✋ 8	☀ 9	☾ 0	☾ 1	☾ 2	☾ 3	☾ 4	☾ 5
♁	9 6	♁ 7	♁ 8	♁ 9	♁ 0	♁ 1	♁ 2	♁ 3	♁ 4	♁ 5	♁ 6	♁ 7	♁ 8	♁ 9	♁ 0	♁ 1
◻	1 2	◻ 3	◻ 4	◆ 5	◆ 6	◆ 7	◆ 8	◆ 9	⊠ 0	⊠ 1	⊠ 2	⊠ 3	⊠ 4	“ 5	” 6	◻ 7
⊙	1 8	① 2	② 3	③ 4	④ 5	⑤ 6	⑥ 7	⑦ 8	⑧ 9	⑨ 0	⑩ 1	⓪ 2	① 3	② 4	③ 5	④ 6
⑤	4 4	⑥ 5	⑦ 6	⑧ 7	⑨ 8	⑩ 9	⓪ 0	① 1	② 2	③ 3	④ 4	⑤ 5	⑥ 6	⑦ 7	⑧ 8	⑨ 9
·	6 0	○ 1	● 2	● 3	◎ 4	◎ 5	○ 6	▪ 7	◻ 8	▲ 9	✦ 0	★ 1	★ 2	7 3	✳ 4	✳ 5
⊕	7 6	⊕ 7	✧ 8	✧ 9	⊕ 0	⊕ 1	☆ 2	⊕ 3	⊕ 4	⊕ 5	⊕ 6	⊕ 7	⊕ 8	⊕ 9	⊕ 0	⊕ 1
⌚	9 2	⌚ 3	⌚ 4	✋ 5	✋ 6	✋ 7	✋ 8	✋ 9	✋ 0	✋ 1	✋ 2	✋ 3	✋ 4	✋ 5	✋ 6	✋ 7
✋	2 8	✋ 9	✋ 0	✋ 1	✋ 2	✋ 3	✋ 4	✋ 5	✋ 6	✋ 7	✋ 8	✋ 9	✋ 0	✋ 1	✋ 2	✋ 3
➔	2 2	↑ 2	↓ 2	↶ 2	↷ 2	↶ 2	↷ 3	↶ 3	↷ 3	↑ 3	↓ 3	↶ 3	↷ 3	↶ 3	↷ 3	↶ 3

	4		5		6		7		8		9		0		1		2		3		4		5		6		7		8		9
	2		2		2		2		2		2		2		2		2		2		2		2		2		2		2		2
⇒	4	↑	4	↓	4	↔	4	↕	4	↖	4	↗	4	↘	4	□	4	□	5	×	5	✓	5	⊗	5	⊗	5	⊗	5	⊗	5
	0		1		2		3		4		5		6		7		8		9		0		1		2		3		4		5

## 2.9 Набор Web-цветов

Тип color, поддерживающий цветовые константы:

Black	DarkGreen	DarkSlate Gray	Olive	Green	Teal	Navy	Purple
Maroon	Indigo	MidnightBlue	DarkBlue	DarkOlive Green	Saddle Brown	ForestGreen	OliveDrab
SeaGreen	DarkGoldenrod	DarkSlate Blue	Sienna	MediumBlue	Brown	DarkTurquoise	DimGray
LightSeaGreen	DarkViolet	FireBrick	MediumVioletRed	MediumSeaGreen	Chocolate	Crimson	SteelBlue
Goldenrod	MediumSpringGreen	LawnGreen	CadetBlue	DarkOrchid	YellowGreen	LimeGreen	OrangeRed
DarkOrange	Orange	Gold	Yellow	Chartreuse	Lime	SpringGreen	Aqua
DeepSkyBlue	Blue	Magenta	Red	Gray	SlateGray	Peru	BlueViolet
LightSlateGray	DeepPink	MediumTurquoise	DodgerBlue	Turquoise	RoyalBlue	SlateBlue	DarkKhaki
IndianRed	MediumOrchid	GreenYellow	MediumAquaMarine	DarkSeaGreen	Tomato	RosyBrown	Orchid
MediumPurple	PaleVioletRed	Coral	CornflowerBlue	DarkGray	SandyBrown	MediumSlateBlue	Tan
DarkSalmon	BurlyWood	HotPink	Salmon	Violet	LightCoral	SkyBlue	LightSalmon
Plum	Khaki	LightGreen	Aquamarine	Silver	LightSky	LightSteel	LightBlue

		n			yBlue	Blue	
PaleGreen	Thistle	PowderBlue	PaleGoldenrod	PaleTurquoise	LightGray	Wheat	NavajoWhite
Moccasin	LightPink	Gainsboro	PeachPuff	Pink	Bisque	LightGoldenrod	BlanchedAlmond
LemonChiffon	Beige	AntiqueWhite	PapayaWhip	Cornsilk	LightYellow	LightCyan	Linen
Lavender	MistyRose	OldLace	WhiteSmoke	Seashell	Ivory	Honeydew	AliceBlue
LavenderBlush	MintCream	Snow	White				

## 2.10 Линии индикаторов

Идентификаторы линий индикаторов, используемых при вызове функций `iMACD()`, `iRVI()` и `iStochastic()`. Могут иметь одно из следующего значений:

Константа	Значение	Описание
MODE_MAIN	0	Основная линия
MODE_SIGNAL	1	Сигнальная линия

Идентификаторы линий Average Directional Movement Index, используемых при вызове функции `iADX()`:

Константа	Значение	Описание
MODE_MAIN	0	Основная линия
MODE_PLUSDI	1	Линия +DI
MODE_MINUSDI	2	Линия -DI

Идентификаторы линий индикаторов, используемых при вызове функций iBands(), iEnvelopes(), iEnvelopesOnArray(), iFractals() и iGator():

Константа	Значение	Описание
MODE_UPPER	1	Верхняя линия
MODE_LOWER	2	Нижняя линия

## 2.11 Ichimoku Kinko Hyo

Идентификаторы линий индикатора Ichimoku Kinko Hyo, используемых при вызове функции iichimoku(). Могут иметь одно из следующих значений:

Константа	Значение	Описание
MODE_TENKANSEN	1	Tenkan-sen
MODE_KIJUNSEN	2	Kijun-sen
MODE_SENKOUSPANA	3	Senkou Span A
MODE_SENKOUSPANB	4	Senkou Span B
MODE_CHINKOUSPAN	5	Chinkou Span

## 2.12 Методы скользящих

Метод вычисления скользящего среднего (Moving Average). Используется в индикаторах iAlligator(), iEnvelopes(), iEnvelopesOnArray, iForce(), iGator(), iMA(), iMAOnArray(), iStdDev(), iStdDevOnArray() и iStochastic(). Может быть одним из следующих:

Константа	Значение	Описание
MODE_SMA	0	Простое скользящее среднее
MODE_EMA	1	Экспоненциальное скользящее среднее
MODE_SMMA	2	Сглаженное скользящее среднее
MODE_LWMA	3	Линейно-взвешенное скользящее среднее

## 2.13 MessageBox

Коды возврата функции `MessageBox()`. Если окно сообщения имеет кнопку *Отмена (Cancel)*, то функция возвращает значение IDCANCEL при нажатой клавише ESC или кнопке *Отмена (Cancel)*. Если окно сообщения не имеет кнопки *Отмена (Cancel)*, нажатие ESC не дает никакого эффекта.

Константа	Значение	Описание
IDOK	1	Выбрана кнопка <i>ОК</i>
IDCANCEL	2	Выбрана кнопка <i>Отмена (Cancel)</i>
IDABORT	3	Выбрана кнопка <i>Прервать (Abort)</i>
IDRETRY	4	Выбрана кнопка <i>Повтор (Retry)</i>
IDIGNORE	5	Выбрана кнопка <i>Пропустить (Ignore)</i>
IDYES	6	Выбрана кнопка <i>Да (Yes)</i>
IDNO	7	Выбрана кнопка <i>Нет (No)</i>
IDTRYAGAIN	10	Выбрана кнопка <i>Повторить (Try Again)</i>
IDCONTINUE	11	Выбрана кнопка <i>Продолжить (Continue)</i>

Эти коды возврата определены в файле `WinUser32.mqh`, поэтому необходимо включать этот заголовочный файл в программы через `#include <WinUser32.mqh>`.

Основные флаги функции `MessageBox()` определяют содержание и поведение диалогового окна. Это значение может быть комбинацией флагов из следующих групп флагов:

Константа	Значение	Описание
MB_OK	0x00000000	Окно сообщения содержит одну кнопку: ОК. По умолчанию
MB_OKCANCEL	0x00000001	Окно сообщения содержит две кнопки: ОК и Cancel

MB_ABORTRETRYIGNORE	0x00000002	Окно сообщения содержит три кнопки: Abort, Retry и Ignore
MB_YESNOCANCEL	0x00000003	Окно сообщения содержит три кнопки: Yes, No и Cancel
MB_YESNO	0x00000004	Окно сообщения содержит две кнопки: Yes и No
MB_RETRYCANCEL	0x00000005	Окно сообщения содержит две кнопки: Retry и Cancel
MB_CANCELTRYCONTINUE	0x00000006	Окно сообщения содержит три кнопки: Cancel, Try Again, Continue

Для отображения иконки в окне сообщения необходимо определить дополнительные флаги:

Константа	Значение	Описание
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	0x00000010	Изображение знака STOP
MB_ICONQUESTION	0x00000020	Изображение вопросительного знака
MB_ICONEXCLAMATION, MB_ICONWARNING	0x00000030	Изображение восклицательного знака
MB_ICONINFORMATION, MB_ICONASTERISK	0x00000040	Изображение, состоящее из строчного знака <b>i</b> в круге

Кнопки по умолчанию задаются следующими флагами:

Константа	Значение	Описание
MB_DEFBUTTON1	0x00000000	Первая кнопка MB_DEFBUTTON1 - кнопка выбрана по умолчанию, если MB_DEFBUTTON2,

		MB_DEFBUTTON3, или MB_DEFBUTTON4 не определены
MB_DEFBUTTON2	0x00000100	Вторая кнопка - кнопка по умолчанию
MB_DEFBUTTON3	0x00000200	Третья кнопка - кнопка по умолчанию
MB_DEFBUTTON4	0x00000300	Четвертая кнопка - кнопка по умолчанию

Флаги поведения функции `MessageBox()` определены в файле `WinUser32.mqh`, поэтому необходимо включать этот заголовочный файл в программы через `#include <WinUser32.mqh>`.  
Здесь перечислены не все возможные флаги. Более подробную информацию можно получить в описании Win32 API

## **2.14 Типы объектов**

Идентификаторы типов графических объектов используются в функциях `ObjectCreate()`, `ObjectsDeleteAll()` и `ObjectType()`. Могут быть любыми из следующих величин (объекты могут иметь 1-3 координаты в зависимости от типа):

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
OBJ_VLINE	0	Вертикальная линия. Использует время в качестве первой координаты, цена игнорируется
OBJ_HLINE	1	Горизонтальная линия. Использует цену в качестве первой координаты, время игнорируется
OBJ_TREND	2	Трендовая линия. Использует 2 координаты
OBJ_TRENDBYANGLE	3	Трендовая линия по углу. Использует 2 координаты, либо первую координату и угол. Для установки угла линии (свойство <code>OBJPROP_ANGLE</code> ) используется функция <code>ObjectSet()</code>

OBJ_REGRESSION	4	Канал линейной регрессии. Использует временные составляющие 2 координат, ценовые составляющие игнорируются
OBJ_CHANNEL	5	Равноудаленный канал. Использует 3 координаты
OBJ_STDDEVCHANNEL	6	Канал стандартных отклонений. Использует временные составляющие 2 координат, ценовые составляющие игнорируются
OBJ_GANNLIN	7	Линия Ганна. Использует 2 координаты, но ценовая составляющая второй координаты игнорируется. Для установки соотношения между временной и ценовой шкалами (свойство OBJPROP_SCALE) используется функция <u>ObjectSet()</u>
OBJ_GANNFAN	8	Веер Ганна. Использует 2 координаты, но ценовая составляющая второй координаты игнорируется. Для установки соотношения между временной и ценовой шкалами (свойство OBJPROP_SCALE) используется функция <u>ObjectSet()</u>
OBJ_GANNGRID	9	Сетка Ганна. Использует 2 координаты, но ценовая составляющая второй координаты игнорируется. Для установки соотношения между временной и ценовой шкалами (свойство OBJPROP_SCALE) используется функция <u>ObjectSet()</u>
OBJ_FIBO	10	Уровни Фибоначчи. Использует 2 координаты. Для установки количества уровней (свойство OBJPROP_FIBOLEVELS) и значения уровней (свойство OBJPROP_FIRSTLEVEL+n) используется

		функция <u>ObjectSet()</u>
OBJ_FIBOTIMES	11	Временные зоны Фибоначчи. Использует 2 координаты. Для установки количества уровней (свойство OBJPROP_FIBOLEVELS) и значения уровней (свойство OBJPROP_FIRSTLEVEL+n) используется функция <u>ObjectSet()</u>
OBJ_FIBOFAN	12	Веер Фибоначчи. Использует 2 координаты. Для установки количества уровней (свойство OBJPROP_FIBOLEVELS) и значения уровней (свойство OBJPROP_FIRSTLEVEL+n) используется функция <u>ObjectSet()</u>
OBJ_FIBOARC	13	Дуги Фибоначчи. Использует 2 координаты. Для установки количества уровней (свойство OBJPROP_FIBOLEVELS) и значения уровней (свойство OBJPROP_FIRSTLEVEL+n) используется функция <u>ObjectSet()</u>
OBJ_EXPANSION	14	Расширение Фибоначчи. Использует 3 координаты. Для установки количества уровней (свойство OBJPROP_FIBOLEVELS) и значения уровней (свойство OBJPROP_FIRSTLEVEL+n) используется функция <u>ObjectSet()</u>
OBJ_FIBOCHANNEL	15	Канал Фибоначчи. Использует 3 координаты. Для установки количества уровней (свойство OBJPROP_FIBOLEVELS) и значения уровней (свойство OBJPROP_FIRSTLEVEL+n) используется функция <u>ObjectSet()</u>

OBJ_RECTANGLE	16	Прямоугольник. Использует 2 координаты
OBJ_TRIANGLE	17	Треугольник. Использует 3 координаты
OBJ_ELLIPSE	18	Эллипс. Использует 2 координаты. Для установки соотношения между временной и ценовой шкалами (свойство OBJPROP_SCALE) используется функция <u>ObjectSet()</u>
OBJ_PITCHFORK	19	Вилы Эндрюса. Использует 3 координаты
OBJ_CYCLES	20	Временные ряды (циклические линии). Использует 2 координаты
OBJ_TEXT	21	Текст. Использует 1 координату. Для установки угла выводимого текста (свойство OBJPROP_ANGLE) используется функция <u>ObjectSet()</u> . Для изменения текста используется функция <u>ObjectSetText()</u>
OBJ_ARROW	22	Стрелки (символы). Использует 1 координату. Для установки кода символа (свойство OBJPROP_ARROWCODE) используется функция <u>ObjectSet()</u>
OBJ_LABEL	23	Текстовая метка. Не использует координат. Для установки координат, задаваемых в пикселях относительно угла привязки (свойства OBJPROP_CORNER, OBJPROP_XDISTANCE, OBJPROP_YDISTANCE) используется функция <u>ObjectSet()</u> . Для изменения текста используется функция <u>ObjectSetText()</u>

## 2.15 Свойства объектов

Идентификаторы свойств объекта используются в функциях `ObjectGet()` и `ObjectSet()`. Могут быть любыми из следующих величин:

Константа	Значение	Тип	Описание
OBJPROP_TIME1	0	datetime	Получает/устанавливает первую координату времени
OBJPROP_PRICE1	1	double	Получает/устанавливает первую координату цены
OBJPROP_TIME2	2	datetime	Получает/устанавливает вторую координату времени
OBJPROP_PRICE2	3	double	Получает/устанавливает вторую координату цены
OBJPROP_TIME3	4	datetime	Получает/устанавливает третью координату времени
OBJPROP_PRICE3	5	double	Получает/устанавливает третью координату цены
OBJPROP_COLOR	6	color	Получает/устанавливает <u>цвет</u> объекта
OBJPROP_STYLE	7	int	Получает/устанавливает <u>стиль</u> <u>линии</u> объекта
OBJPROP_WIDTH	8	int	Получает/устанавливает ширину линии объекта
OBJPROP_BACK	9	bool	Получает/устанавливает флаг фонового отображения объекта
OBJPROP_RAY	10	bool	Получает/устанавливает флаг свойства луч для объектов типа OBJ_TREND и ему подобных
OBJPROP_ELLIPSE	11	bool	Получает/устанавливает флаг

			отображения полного эллипса для объекта OBJ_FIBOARC
OBJPROP_SCALE	12	double	Получает/устанавливает значение масштаба объекта
OBJPROP_ANGLE	13	double	Получает/устанавливает значение угла в градусах объекта OBJ_TRENDBYANGLE
OBJPROP_ARROWCODE	14	int	Получает/устанавливает код стрелки объекта OBJ_ARROW. Может быть одним из символов <u>wingdings</u> или один из <u>предопределенных кодов стрелок</u>
OBJPROP_TIMEFRAMES	15	int	Получает/устанавливает свойство отображения объекта на различных периодах. Может быть одним или комбинацией нескольких из <u>констант видимости объекта</u> .
OBJPROP_DEVIATION	16	double	Получает/устанавливает размер отклонения для объекта OBJ_STDDEVCHANNEL
OBJPROP_FONTSIZE	100	int	Получает/устанавливает размер шрифта для объектов OBJ_TEXT и OBJ_LABEL
OBJPROP_CORNER	101	int	Получает/устанавливает номер угла привязки для объекта OBJ_LABEL. Принимает значения 0-3
OBJPROP_XDISTANCE	102	int	Получает/устанавливает

			расстояние X-координаты в пикселях относительно угла привязки для объекта OBJ_LABEL
OBJPROP_YDISTANCE	103	int	Получает/устанавливает расстояние Y-координаты в пикселях относительно угла привязки для объекта OBJ_LABEL
OBJPROP_FIBOLEVELS	200	int	Получает/устанавливает число уровней объекта Fibonacci. Может быть от 1 до 32
OBJPROP_LEVELCOLOR	201	color	Получает/устанавливает <u>цвет</u> линии уровня объекта
OBJPROP_LEVELSTYLE	202	int	Получает/устанавливает стиль линии уровня объекта
OBJPROP_LEVELWIDTH	203	int	Получает/устанавливает ширину линии уровня объекта
OBJPROP_FIRSTLEVEL+n	210+n	int	Получает/устанавливает значения уровня объекта Fibonacci с индексом <i>n</i> . Индекс <i>n</i> может быть от 0 до (количество уровней -1), но не более 31

## 2.16 Видимость объектов

Идентификаторы свойств объекта используются в функциях ObjectGet() и ObjectSet(). Могут быть любыми из следующих величин:

Константа	Значение	Тип	Описание
-----------	----------	-----	----------

OBJPROP_TIME1	0	datetime	Получает/устанавливает первую координату времени
OBJPROP_PRICE1	1	double	Получает/устанавливает первую координату цены
OBJPROP_TIME2	2	datetime	Получает/устанавливает вторую координату времени
OBJPROP_PRICE2	3	double	Получает/устанавливает вторую координату цены
OBJPROP_TIME3	4	datetime	Получает/устанавливает третью координату времени
OBJPROP_PRICE3	5	double	Получает/устанавливает третью координату цены
OBJPROP_COLOR	6	color	Получает/устанавливает <u>цвет</u> объекта
OBJPROP_STYLE	7	int	Получает/устанавливает <u>стиль</u> <u>линии</u> объекта
OBJPROP_WIDTH	8	int	Получает/устанавливает ширину линии объекта
OBJPROP_BACK	9	bool	Получает/устанавливает флаг фонового отображения объекта
OBJPROP_RAY	10	bool	Получает/устанавливает флаг свойства луч для объектов типа OBJ_TREND и ему подобных
OBJPROP_ELLIPSE	11	bool	Получает/устанавливает флаг отображения полного эллипса для объекта OBJ_FIBOARC
OBJPROP_SCALE	12	double	Получает/устанавливает значение масштаба объекта

OBJPROP_ANGLE	13	double	Получает/устанавливает значение угла в градусах объекта OBJ_TRENDBYANGLE
OBJPROP_ARROWCODE	14	int	Получает/устанавливает код стрелки объекта OBJ_ARROW. Может быть одним из символов <u>wingdings</u> или один из <u>предопределенных кодов стрелок</u>
OBJPROP_TIMEFRAMES	15	int	Получает/устанавливает свойство отображения объекта на различных периодах. Может быть одним или комбинацией нескольких из <u>констант видимости объекта</u> .
OBJPROP_DEVIATION	16	double	Получает/устанавливает размер отклонения для объекта OBJ_STDDEVCHANNEL
OBJPROP_FONTSIZE	100	int	Получает/устанавливает размер шрифта для объектов OBJ_TEXT и OBJ_LABEL
OBJPROP_CORNER	101	int	Получает/устанавливает номер угла привязки для объекта OBJ_LABEL. Принимает значения 0-3
OBJPROP_XDISTANCE	102	int	Получает/устанавливает расстояние X-координаты в пикселях относительно угла привязки для объекта OBJ_LABEL

OBJPROP_YDISTANCE	103	int	Получает/устанавливает расстояние Y-координаты в пикселях относительно угла привязки для объекта OBJ_LABEL
OBJPROP_FIBOLEVELS	200	int	Получает/устанавливает число уровней объекта Fibonacci. Может быть от 1 до 32
OBJPROP_LEVELCOLOR	201	color	Получает/устанавливает <u>цвет</u> линии уровня объекта
OBJPROP_LEVELSTYLE	202	int	Получает/устанавливает стиль линии уровня объекта
OBJPROP_LEVELWIDTH	203	int	Получает/устанавливает ширину линии уровня объекта
OBJPROP_FIRSTLEVEL+n	210+n	int	Получает/устанавливает значения уровня объекта Fibonacci с индексом <i>n</i> . Индекс <i>n</i> может быть от 0 до (количество уровней -1), но не более 31

## **2.17 Причины деинициализации**

Коды причины деинициализации, возвращаемые функцией `UninitializeReason()`. Могут иметь любые из следующих значений:

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
	0	Скрипт самостоятельно завершил свою работу
REASON_REMOVE	1	Программа удалена с графика

REASON_RECOMPILE	2	Программа перекомпилирована
REASON_CHARTCHANGE	3	Символ или период графика был изменен
REASON_CHARTCLOSE	4	График закрыт
REASON_PARAMETERS	5	Входные параметры были изменены пользователем
REASON_ACCOUNT	6	Активирован другой счет

## **2.18 Специальные константы**

Специальные константы, используемые для указания состояния параметров и переменных. Могут быть следующими величинами:

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
NULL	0	Указывает пустое состояние строки
EMPTY	-1	Указывает пустое состояние параметра
EMPTY_VALUE	0x7FFFFFFF	Значение по умолчанию, указатель пустого значения. Используется в <u>пользовательских индикаторах</u>
CLR_NONE	0xFFFFFFFF	Указывает отсутствие цвета
WHOLE_ARRAY	0	Используется с <u>функциями массивов</u> . Указывает, что все элементы массива должны быть обработаны

## **2.18 Коды ошибок**

`GetLastError()` - функция, возвращающая коды ошибок. Кодовые константы ошибок определены в файле `stderr.mqh`. Для вывода текстовых сообщений следует использовать функцию `ErrorDescription()`, определенную в файле `stdlib.mqh`.

**Пример:**

```

#include <stderror.mqh>
#include <stdlib.mqh>
void SendMyMessage(string text)
{
    int check;
    SendMail("Test", text);
    check=GetLastError();
    if(check!=ERR_NO_ERROR) Print("Сообщение не отправлено. Ошибка:
",ErrorDescription(check));
}

```

Коды ошибок, возвращаемые торговым сервером:

Константа	Значение	Описание
ERR_NO_ERROR	0	Нет ошибки
ERR_NO_RESULT	1	Нет ошибки, но результат неизвестен
ERR_COMMON_ERROR	2	Общая ошибка
ERR_INVALID_TRADE_PARAMETERS	3	Неправильные параметры
ERR_SERVER_BUSY	4	Торговый сервер занят
ERR_OLD_VERSION	5	Старая версия клиентского терминала
ERR_NO_CONNECTION	6	Нет связи с торговым сервером
ERR_NOT_ENOUGH_RIGHTS	7	Недостаточно прав
ERR_TOO_FREQUENT_REQUESTS	8	Слишком частые запросы
ERR_MALFUNCTIONAL_TRADE	9	Недопустимая операция нарушающая функционирование сервера
ERR_ACCOUNT_DISABLED	64	Счет заблокирован
ERR_INVALID_ACCOUNT	65	Неправильный номер

		счета
ERR_TRADE_TIMEOUT	128	Истек срок ожидания совершения сделки
ERR_INVALID_PRICE	129	Неправильная цена
ERR_INVALID_STOPS	130	Неправильные стопы
ERR_INVALID_TRADE_VOLUME	131	Неправильный объем
ERR_MARKET_CLOSED	132	Рынок закрыт
ERR_TRADE_DISABLED	133	Торговля запрещена
ERR_NOT_ENOUGH_MONEY	134	Недостаточно денег для совершения операции
ERR_PRICE_CHANGED	135	Цена изменилась
ERR_OFF_QUOTES	136	Нет цен
ERR_BROKER_BUSY	137	Брокер занят
ERR_REQUOTE	138	Новые цены
ERR_ORDER_LOCKED	139	Ордер заблокирован и уже обрабатывается
ERR_LONG_POSITIONS_ONLY_ALLOWED	140	Разрешена только покупка
ERR_TOO_MANY_REQUESTS	141	Слишком много запросов
ERR_TRADE_MODIFY_DENIED	145	Модификация запрещена, так как ордер слишком близок к рынку
ERR_TRADE_CONTEXT_BUSY	146	Подсистема торговли занята
ERR_TRADE_EXPIRATION_DENIED	147	Использование даты истечения ордера запрещено брокером

ERR_TRADE_TOO_MANY_ORDERS	148	Количество открытых и отложенных ордеров достигло предела, установленного брокером.
ERR_TRADE_HEDGE_PROHIBITED	149	Попытка открыть противоположную позицию к уже существующей в случае, если хеджирование запрещено.
ERR_TRADE_PROHIBITED_BY_FIFO	150	Попытка закрыть позицию по инструменту в противоречии с правилом FIFO.

Коды ошибок выполнения MQL4-программы:

Константа	Значение	Описание
ERR_NO_MQLERROR	4000	Нет ошибки
ERR_WRONG_FUNCTION_POINTER	4001	Неправильный указатель функции
ERR_ARRAY_INDEX_OUT_OF_RANGE	4002	Индекс массива - вне диапазона
ERR_NO_MEMORY_FOR_CALL_STACK	4003	Нет памяти для стека функций
ERR_RECURSIVE_STACK_OVERFLOW	4004	Переполнение стека после рекурсивного вызова
ERR_NOT_ENOUGH_STACK_FOR_PARAM	4005	На стеке нет памяти для передачи параметров

ERR_NO_MEMORY_FOR_PARAM_STRING	4006	Нет памяти для строкового параметра
ERR_NO_MEMORY_FOR_TEMP_STRING	4007	Нет памяти для временной строки
ERR_NOT_INITIALIZED_STRING	4008	Неинициализированная строка
ERR_NOT_INITIALIZED_ARRAYSTRING	4009	Неинициализированная строка в массиве
ERR_NO_MEMORY_FOR_ARRAYSTRING	4010	Нет памяти для строкового массива
ERR_TOO_LONG_STRING	4011	Слишком длинная строка
ERR_REMAINDER_FROM_ZERO_DIVIDE	4012	Остаток от деления на ноль
ERR_ZERO_DIVIDE	4013	Деление на ноль
ERR_UNKNOWN_COMMAND	4014	Неизвестная команда
ERR_WRONG_JUMP	4015	Неправильный переход
ERR_NOT_INITIALIZED_ARRAY	4016	Неинициализированный массив
ERR_DLL_CALLS_NOT_ALLOWED	4017	Вызовы DLL не разрешены
ERR_CANNOT_LOAD_LIBRARY	4018	Невозможно загрузить библиотеку
ERR_CANNOT_CALL_FUNCTION	4019	Невозможно вызвать функцию
ERR_EXTERNAL_CALLS_NOT_ALLOWED	4020	Вызовы внешних библиотечных функций не разрешены
ERR_NO_MEMORY_FOR_RETURNED_STR	4021	Недостаточно памяти для

		строки, возвращаемой из функции
ERR_SYSTEM_BUSY	4022	Система занята
ERR_INVALID_FUNCTION_PARAMSCNT	4050	Неправильное количество параметров функции
ERR_INVALID_FUNCTION_PARAMVALUE	4051	Недопустимое значение параметра функции
ERR_STRING_FUNCTION_INTERNAL	4052	Внутренняя ошибка строковой функции
ERR_SOME_ARRAY_ERROR	4053	Ошибка массива
ERR_INCORRECT_SERIESARRAY_USING	4054	Неправильное использование массива-таймсерии
ERR_CUSTOM_INDICATOR_ERROR	4055	Ошибка пользовательского индикатора
ERR_INCOMPATIBLE_ARRAYS	4056	Массивы несовместимы
ERR_GLOBAL_VARIABLES_PROCESSING	4057	Ошибка обработки глобальных переменных
ERR_GLOBAL_VARIABLE_NOT_FOUND	4058	Глобальная переменная не обнаружена
ERR_FUNC_NOT_ALLOWED_IN_TESTING	4059	Функция не разрешена в тестовом режиме
ERR_FUNCTION_NOT_CONFIRMED	4060	Функция не разрешена
ERR_SEND_MAIL_ERROR	4061	Ошибка отправки почты
ERR_STRING_PARAMETER_EXPECTED	4062	Ожидается параметр типа string
ERR_INTEGER_PARAMETER_EXPECTED	4063	Ожидается параметр типа

		integer
ERR_DOUBLE_PARAMETER_EXPECTED	4064	Ожидается параметр типа double
ERR_ARRAY_AS_PARAMETER_EXPECTED	4065	В качестве параметра ожидается массив
ERR_HISTORY_WILL_UPDATED	4066	Запрошенные исторические данные в состоянии обновления
ERR_TRADE_ERROR	4067	Ошибка при выполнении торговой операции
ERR_END_OF_FILE	4099	Конец файла
ERR_SOME_FILE_ERROR	4100	Ошибка при работе с файлом
ERR_WRONG_FILE_NAME	4101	Неправильное имя файла
ERR_TOO_MANY_OPENED_FILES	4102	Слишком много открытых файлов
ERR_CANNOT_OPEN_FILE	4103	Невозможно открыть файл
ERR_INCOMPATIBLE_FILEACCESS	4104	Несовместимый режим доступа к файлу
ERR_NO_ORDER_SELECTED	4105	Ни один ордер не выбран
ERR_UNKNOWN_SYMBOL	4106	Неизвестный символ
ERR_INVALID_PRICE_PARAM	4107	Неправильный параметр цены для торговой функции
ERR_INVALID_TICKET	4108	Неверный номер тикета
ERR_TRADE_NOT_ALLOWED	4109	Торговля не разрешена. Необходимо включить

		опцию "Разрешить советнику торговать" в свойствах эксперта.
ERR_LONGS_NOT_ALLOWED	4110	Длинные позиции не разрешены. Необходимо проверить свойства эксперта.
ERR_SHORTS_NOT_ALLOWED	4111	Короткие позиции не разрешены. Необходимо проверить свойства эксперта.
ERR_OBJECT_ALREADY_EXISTS	4200	Объект уже существует
ERR_UNKNOWN_OBJECT_PROPERTY	4201	Запрошено неизвестное свойство объекта
ERR_OBJECT_DOES_NOT_EXIST	4202	Объект не существует
ERR_UNKNOWN_OBJECT_TYPE	4203	Неизвестный тип объекта
ERR_NO_OBJECT_NAME	4204	Нет имени объекта
ERR_OBJECT_COORDINATES_ERROR	4205	Ошибка координат объекта
ERR_NO_SPECIFIED_SUBWINDOW	4206	Не найдено указанное подокно
ERR_SOME_OBJECT_ERROR	4207	Ошибка при работе с объектом

### **3 Выполнение программ**

Для того, чтобы MQL4-программа могла работать, она должна быть откомпилирована (кнопка "Компилировать" или клавиша F5). Компиляция должна пройти без ошибок (допускаются предупреждения, которые необходимо проанализировать). При этом в соответствующем директории, *terminal\_dir\experts*, *terminal\_dir\experts\indicators* или *terminal\_dir\experts\scripts*, должен быть создан выполняемый файл с тем же именем и расширением EX4. Именно этот файл может быть запущен на выполнение.

Эксперты, пользовательские индикаторы и скрипты прикрепляются к одному из открытых графиков путем перетаскивания мышью из окна "Навигатор" клиентского терминала на соответствующий график (технология Drag'n'Drop). MQL4-программы могут работать только при включенном клиентском терминале.

Для того, чтобы эксперт прекратил работать, его необходимо удалить с графика при помощи выбора "Советники - Удалить" из контекстного меню графика. На работу советника также влияет состояние кнопки "Разрешить/запретить советников".

Для того, чтобы пользовательский индикатор прекратил работу, его необходимо удалить с графика.

Пользовательские индикаторы и советники работают до тех пор пока их явно не удалят с графика, информация о прикрепленных советниках и пользовательских индикаторах сохраняется между запусками клиентского терминала. Скрипты выполняются однократно и удаляются автоматически по завершению своей работы либо по закрытию или изменению состояния текущего графика, либо по завершению работы клиентского терминала. При повторном запуске клиентского терминала скрипты не запускаются, так как информация о них не сохраняется.

На одном графике могут работать максимум по одному эксперту и скрипту и неограниченное количество индикаторов

#### **3.1 Выполнение программ**

Сразу же после присоединения к графику программа начинает работу с функции `init()`. Функция `init()` присоединенного к графику советника или пользовательского индикатора запускается также сразу после старта клиентского терминала и подгрузки (это касается

только советников и не касается индикаторов) исторических данных, после смены финансового инструмента и/или периода графика, после перекомпиляции программы в редакторе MetaEditor, после смены входных параметров из окна настройки эксперта или пользовательского индикатора. Советник также инициализируется после смены счёта

Каждая присоединенная к графику программа завершает работу функцией `deinit()`. Функция `deinit()` запускается также при завершении работы клиентского терминала, при закрытии графика, непосредственно перед сменой финансового инструмента и/или периода графика, при удачной перекомпиляции программы, при смене входных параметров, а также при смене счета. Причину деинициализации можно получить, используя функцию `UninitializeReason()` при выполнении функции `deinit()`. Выполнение функции `deinit()` ограничивается 2.5 секундами. Если за это время функция не закончила свою работу, то ее выполнение завершается принудительно. Исключения составляют скрипты, которые как правило, самостоятельно завершают свою работу без команды `извне`. Если же скрипт работает долго (например, по причине бесконечного цикла), то его можно завершить командой `извне` (при удалении скрипта из контекстного меню графика, при присоединении к графику нового скрипта, при закрытии графика, при смене финансового инструмента и/или периода графика). В этом случае `deinit()` также ограничивается 2.5 секундами.

При поступлении новых котировок выполняется функция `start()` у присоединенных советников и пользовательских индикаторов. Если при поступлении новой котировки выполнялась функция `start()`, запущенная на предыдущей котировке, то пришедшая котировка будет проигнорирована советником. Все пришедшие во время выполнения программы новые котировки программой игнорируются до тех пор, пока не завершится очередное выполнение функции `start()`. После этого функция `start()` будет запущена только после прихода очередной новой котировки. У пользовательских индикаторов функция `start()` запускается для пересчета также после смены символа или периода текущего графика вне зависимости от поступления новых котировок. Функция `start()` не запускается при включенном запрете использования советников (кнопка "Разрешить/запретить советников"). Однако запрещение работы советников путем нажатия на указанную кнопку не прерывает текущее выполнение функции `start()`. Функция `start()` не запускается на выполнение при открытом окне свойств советника. Окно свойств не может быть открыто в момент выполнения советника.

Отсоединение программы от графика, смена финансового инструмента и/или периода графика, смена счета, закрытие графика, а также завершение работы клиентского терминала прерывает выполнение программы. Если функция `start()` выполнялась на момент команды на завершение работы, оставшееся время работы ограничивается 2.5 секундами. Программа может узнать, что её пытаются завершить при помощи встроенной функции `IsStopped()` и корректно закончить свою работу.

Выполнение скриптов не зависит от приходящих котировок. При смене финансового инструмента и/или периода графика скрипт завершает свою работу и выгружается из клиентского терминала.

Скрипты и эксперты работают в собственном потоке. Пользовательские индикаторы работают в интерфейсном потоке. Если же пользовательский индикатор вызван при помощи функции `iCustom()`, то этот индикатор работает в потоке вызвавшей его программы. Библиотечные (импортируемые) функции также работают в потоке вызывающей программы

### **3.2 Вызов импортируемых функций**

Для импорта функций во время выполнения mql4-программы используется так называемое позднее связывание. Это значит, что пока не вызвана импортируемая функция, соответствующий модуль (ex4 или dll) не загружается. Библиотеки MQL4 и DLL выполняются в потоке вызывающего модуля.

Не рекомендуется использовать полностью квалифицированное имя загружаемого модуля вида *Drive:\Directory\FileName.Ext*. Библиотеки MQL4 загружаются из папки *terminal\_dir\experts\libraries*. Если библиотека не была найдена, то производится попытка загрузить библиотеку из папки *terminal\_dir\experts*.

Системные библиотеки (DLL) загружаются по правилам операционной системы. Если библиотека уже загружена (например, другим экспертом и даже из другого клиентского терминала, запущенного параллельно), то обращение идет к уже загруженной библиотеке.

В противном случае поиск идет в следующей последовательности:

1. Директория *terminal\_dir\experts\libraries*.
2. Директория, из которой запущен клиентский терминал *terminal\_dir*.
3. Текущая директория.
4. Системная директория *windows\_dir\SYSTEM32* ( или *windows\_dir\SYSTEM* для Win98).
5. Директория, в которую установлена операционная система *windows\_dir*.

## 6. Директории, перечисленные в системной переменной окружения PATH.

Если библиотека DLL использует в своей работе другую DLL, то в случае отсутствия второй DLL первая не сможет загрузиться.

В отличие от системных библиотек пользовательские библиотеки (MQL4) загружаются для каждого вызывающего модуля отдельно, независимо от того, была ли загружена вызываемая библиотека каким-либо другим модулем. Например, модуль caller.ex4 вызывает функции из библиотеки lib1.ex4 и lib2.ex4. В свою очередь, библиотека lib1.ex4 вызывает функции из библиотеки lib2.ex4. В этом случае будет загружена одна копия библиотеки lib1.ex4 и две копии библиотеки lib2.ex4, несмотря на то, что все вызовы исходят из модуля caller.ex4.

Функции, импортируемые из DLL в mql4-программу, должны обеспечивать соглашение о связях, принятое для функций Windows API. Для обеспечения такого соглашения в исходном тексте программ, написанных на языках C или C++ используется ключевое слово `__stdcall`, которое является специфическим для компиляторов от фирмы Microsoft(r).

Обсуждаемое соглашение о связях характеризуется следующим:

- вызывающая функция (в нашем случае mql4-программа) должна "видеть" прототип вызываемой (импортируемой из DLL) функции, для того чтобы правильно сложить параметры на стек;
- вызывающая функция (в нашем случае mql4-программа) складывает параметры на стек в обратном порядке, справа налево - именно в таком порядке импортируемая функция считывает переданные ей параметры;
- параметры передаются по значению, за исключением тех, которые явно передаются по ссылке (в нашем случае строк)
- импортируемая функция, считывая переданные ей параметры, сама очищает стек.

При описании прототипа импортируемой функции использовать параметры со значениями по умолчанию бесполезно, т.к. все параметры в импортируемую функцию следует передавать явно.

В случае если вызов импортируемой функции оказался неудачным (в настройках эксперта запрещено использовать импортируемые функции либо соответствующая библиотека не смогла загрузиться), эксперт останавливает свою работу с соответствующим сообщением

"expert stopped" в журнале. При этом эксперт не будет запускаться, пока не будет заново проинициализирован. Эксперт может быть переинициализирован в результате перекомпиляции либо после открытия таблицы свойств эксперта и нажатия кнопки ОК

### **3.3 Ошибки выполнения**

В исполняющей подсистеме клиентского терминала существует возможность сохранения кода ошибки в случае ее возникновения при выполнении mql4-программы. Для каждой исполняемой mql4-программы предусмотрена своя собственная специальная переменная last\_error. Перед запуском функции init переменная last\_error обнуляется. При возникновении ошибочной ситуации во время вычислений или в процессе вызова встроенной функции переменная last\_error принимает соответствующий код ошибки. Значение, сохраненное в этой переменной, можно получить при помощи функции GetLastError(). При этом переменная last\_error обнуляется.

Существует ряд критических ошибок, при возникновении которых выполнение программы немедленно прерывается:

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
ERR_WRONG_FUNCTION_POINTER	4001	При вызове внутренней функции обнаружен неправильный указатель вызываемой функции
ERR_NO_MEMORY_FOR_CALL_STACK	4003	При вызове внутренней функции невозможно перераспределить память для стека вызовов функции
ERR_RECURSIVE_STACK_OVERFLOW	4004	При рекурсивном вызове функции исчерпан стек данных
ERR_NO_MEMORY_FOR_PARAM_STRING	4006	При вызове внутренней функции невозможно распределить память для

		передачи строки в качестве параметра функции
ERR_NO_MEMORY_FOR_TEMP_STRING	4007	Невозможно распределить временный буфер для операций со строками
ERR_NO_MEMORY_FOR_ARRAYSTRING	4010	При присвоении невозможно перераспределить память для строки в массиве
ERR_TOO_LONG_STRING	4011	При присвоении слишком длинная результирующая строка для помещения в служебный буфер (невозможно перераспределить память для служебного буфера)
ERR_REMAINDER_FROM_ZERO_DIVIDE	4012	Деление на 0 при взятии остатка от деления
ERR_ZERO_DIVIDE	4013	Деление на 0
ERR_UNKNOWN_COMMAND	4014	Неизвестная инструкция

Если работа программы была прервана из-за критической ошибки, то при следующем запуске функции start или функции deinit можно прочитать код этой ошибки, используя функцию `GetLastError()`. Перед запуском функций start и deinit переменная last\_error не сбрасывается.

Существует ряд критических ошибок, связанных с вызовом импортируемых функций, при возникновении которых выполнение эксперта или индикатора немедленно прерывается и функция start не запускается до тех пор, пока не будет произведена переинициализация эксперта или индикатора.

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
ERR_CANNOT_LOAD_LIBRARY	4018	При вызове импортируемой функции произошла ошибка загрузки dll или ex4-библиотеки
ERR_CANNOT_CALL_FUNCTION	4019	При вызове импортируемой из функции выяснилось, что dll или ex4-библиотека не содержит эту функцию
ERR_DLL_CALLS_NOT_ALLOWED	4017	При вызове импортируемой из dll функции выяснилось, что установлен запрет на вызовы из dll
ERR_EXTERNAL_CALLS_NOT_ALLOWED	4020	При вызове импортируемой из ex4-библиотеки выяснилось, что установлен запрет на вызовы из ex4
ERR_FUNCTION_NOT_CONFIRMED	4060	Не подтвержден вызов импортируемой из dll функции. Если нажата кнопка "Нет", то выполнение программы не прерывается. Если нажата кнопка "Отмена", то выполнение программы немедленно прерывается

Остальные ошибки не прерывают работу программы.

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
ERR_ARRAY_INDEX_OUT_OF_RANGE	4002	Обращение к элементу массива, номер которого находится за пределами массива
ERR_NOT_INITIALIZED_STRING	4008	Неинициализированная строка; строке, участвующей в каком-либо выражении, не присвоено никакого значения
ERR_NOT_INITIALIZED_ARRAYSTRING	4009	Неинициализированная строка в массиве строк; строке, участвующей в каком-либо выражении, не присвоено никакого значения
ERR_NO_MEMORY_FOR_RETURNED_STR	4021	Невозможно перераспределить память для строки, возвращаемой из функции

Никогда не генерируется код ERR\_NO\_MQLERROR (4000).

Существует ряд ошибок, которые возможны только в результате сбоя. При регулярном проявлении какой-либо из нижеперечисленных ошибок необходимо связаться с разработчиками.

<b>Константа</b>	<b>Значение</b>	<b>Описание</b>
ERR_WRONG_FUNCTION_POINTER	4001	При вызове внутренней

		функции обнаружен неправильный указатель вызываемой функции
ERR_UNKNOWN_COMMAND	4014	Неизвестная инструкция
ERR_NOT_INITIALIZED_ARRAY	4016	Неинициализированный массив
ERR_INVALID_FUNCTION_PARAMSCNT	4050	Неверное количество параметров, передаваемых во встроенную функцию
ERR_STRING_FUNCTION_INTERNAL	4052	Ошибка при работе со строками
ERR_TRADE_ERROR	4067	Ошибка при выполнении торговой функции
ERR_SOME_OBJECT_ERROR	4207	Ошибка при работе с объектами

Существует несколько встроенных функций, которые всегда меняют значение переменной last\_error.

<b>Функция</b>	<b>Коды ошибок</b>
<u>AccountFreeMarginCheck</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_UNKNOWN_SYMBOL (4106), ERR_NOT_ENOUGH_MONEY (134)
<u>OrderSend</u>	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_PRICE_PARAM (4107),

	ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), ERR_LONGS_NOT_ALLOWED (4110), ERR_SHORTS_NOT_ALLOWED (4111), <u>коды от торгового сервера</u>
<u>OrderClose</u>	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_PRICE_PARAM (4107), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), <u>коды от торгового сервера</u>
<u>OrderCloseBy</u>	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), <u>коды от торгового сервера</u>
<u>OrderDelete</u>	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), <u>коды от торгового сервера</u>
<u>OrderModify</u>	ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INVALID_PRICE_PARAM (4107), ERR_INVALID_TICKET (4108), ERR_UNKNOWN_SYMBOL (4106), ERR_TRADE_NOT_ALLOWED (4109), <u>коды от торгового сервера</u>
<u>GetLastError</u>	ERR_NO_ERROR (0)

Некоторые функции меняют значение переменной last\_error только в случае возникновения какой-либо ошибки.

Функция	Коды ошибок
<u>ArrayBsearch</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>ArrayCopy</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCOMPATIBLE_ARRAYS (4056), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>ArrayCopyRates</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCOMPATIBLE_ARRAYS (4056), ERR_STRING_PARAMETER_EXPECTED (4062),
<u>ArrayCopySeries</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCORRECT_SERIESARRAY_USING (4054), ERR_INCOMPATIBLE_ARRAYS (4056), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_HISTORY_WILL_UPDATED (4066), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>ArrayDimension</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
<u>ArrayGetAsSeries</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
<u>ArrayInitialize</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>ArrayIsSeries</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065),

	ERR_SOME_ARRAY_ERROR (4053)
<u>ArrayMaximum</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>ArrayMinimum</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>ArrayRange</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INTEGER_PARAMETER_EXPECTED (4063), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>ArrayResize</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>ArraySetAsSeries</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
<u>ArraySize</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
<u>ArraySort</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053), ERR_INCORRECT_SERIESARRAY_USING (4054), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>FileClose</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>FileDelete</u>	ERR_WRONG_FILE_NAME (4101), ERR_SOME_FILE_ERROR (4100)
<u>FileFlush</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>FileIsEnding</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>FileIsLineEnding</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)

<u>FileOpen</u>	ERR_TOO_MANY_OPENED_FILES (4102), ERR_WRONG_FILE_NAME (4101), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_SOME_FILE_ERROR (4100), ERR_CANNOT_OPEN_FILE (4103)
<u>FileOpenHistory</u>	ERR_TOO_MANY_OPENED_FILES (4102), ERR_WRONG_FILE_NAME (4101), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_SOME_FILE_ERROR (4100), ERR_CANNOT_OPEN_FILE (4103)
<u>FileReadArray</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_ARRAY_ERROR (4053), ERR_SOME_FILE_ERROR (4100), ERR_END_OF_FILE (4099)
<u>FileReadDouble</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_END_OF_FILE (4099)
<u>FileReadInteger</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_END_OF_FILE (4099)
<u>FileReadNumber</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), ERR_END_OF_FILE (4099)
<u>FileReadString</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), ERR_TOO_LONG_STRING (4011), ERR_END_OF_FILE (4099)

<u>FileSeek</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>FileSize</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>FileTell</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>FileWrite</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_SOME_FILE_ERROR (4100)
<u>FileWriteDouble</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100)
<u>FileWriteInteger</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100)
<u>FileWriteString</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100), ERR_STRING_PARAMETER_EXPECTED (4062)
<u>FileWriteArray</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCOMPATIBLE_FILEACCESS (4104), ERR_SOME_FILE_ERROR (4100),
<u>GlobalVariableCheck</u>	ERR_STRING_PARAMETER_EXPECTED (4062)
<u>GlobalVariableDel</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLES_PROCESSING (4057)
<u>GlobalVariableGet</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLE_NOT_FOUND (4058)
<u>GlobalVariablesDeleteAll</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLES_PROCESSING (4057)
<u>GlobalVariableSet</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_GLOBAL_VARIABLES_PROCESSING (4057)
<u>GlobalVariableSetOnCondition</u>	ERR_STRING_PARAMETER_EXPECTED (4062),

	ERR_GLOBAL_VARIABLE_NOT_FOUND (4058)
<u>iCustom</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>технические индикаторы,</u> <u>функции доступа к</u> <u>таймсериям</u>	ERR_HISTORY_WILL_UPDATED (4066)
<u>технические индикаторы</u> <u>OnArray</u>	ERR_ARRAY_AS_PARAMETER_EXPECTED (4065), ERR_SOME_ARRAY_ERROR (4053)
<u>IndicatorBuffers</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>IndicatorDigits</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>IndicatorShortName</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>MarketInfo</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_UNKNOWN_SYMBOL (4106), ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>MathArccos</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>MathArcsin</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>MathMod</u>	ERR_ZERO_DIVIDE (4013)
<u>MathSqrt</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>MessageBox</u>	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_STRING_PARAMETER_EXPECTED (4062)
<u>ObjectCreate</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_UNKNOWN_OBJECT_TYPE (4203), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_OBJECT_ALREADY_EXISTS (4200),

	ERR_NO_SPECIFIED_SUBWINDOW (4206)
<u>ObjectDelete</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202)
<u>ObjectDescription</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202)
<u>ObjectFind</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204)
<u>ObjectGet</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_PROPERTY (4201)
<u>ObjectGetFiboDescription</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_TYPE (4203), ERR_UNKNOWN_OBJECT_PROPERTY (4201)
<u>ObjectGetShiftByValue</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_OBJECT_COORDINATES_ERROR (4205)
<u>ObjectGetValueByShift</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_OBJECT_COORDINATES_ERROR (4205)
<u>ObjectMove</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_INVALID_FUNCTION_PARAMVALUE (4051),

	ERR_OBJECT_DOES_NOT_EXIST (4202)
<u>ObjectName</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_ARRAY_INDEX_OUT_OF_RANGE (4002)
<u>ObjectSet</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_PROPERTY (4201)
<u>ObjectSetText</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202)
<u>ObjectSetFiboDescription</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_OBJECT_DOES_NOT_EXIST (4202), ERR_UNKNOWN_OBJECT_TYPE (4203), ERR_UNKNOWN_OBJECT_PROPERTY (4201)
<u>ObjectType</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NO_OBJECT_NAME (4204), ERR_OBJECT_DOES_NOT_EXIST (4202)
<u>OrderClosePrice</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderCloseTime</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderComment</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderCommission</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderExpiration</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderLots</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderMagicNumber</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderOpenPrice</u>	ERR_NO_ORDER_SELECTED (4105)

<u>OrderOpenTime</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderPrint</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderProfit</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderStopLoss</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderSwap</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderSymbol</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderTakeProfit</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderTicket</u>	ERR_NO_ORDER_SELECTED (4105)
<u>OrderType</u>	ERR_NO_ORDER_SELECTED (4105)
<u>PlaySound</u>	ERR_WRONG_FILE_NAME (4101)
<u>SendFTP</u>	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_CUSTOM_INDICATOR_ERROR (4055), ERR_STRING_PARAMETER_EXPECTED (4062)
<u>SendMail</u>	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_FUNCTION_NOT_CONFIRMED (4060), ERR_SEND_MAIL_ERROR (4061)
<u>SetIndexArrow</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>SetIndexBuffer</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_INCORRECT_SERIESARRAY_USING (4054), ERR_INCOMPATIBLE_ARRAYS (4056)
<u>SetIndexDrawBegin</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>SetIndexEmptyValue</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>SetIndexLabel</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_STRING_PARAMETER_EXPECTED (4062)
<u>SetIndexShift</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>SetIndexStyle</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)

<u>SetLevelValue</u>	ERR_INVALID_FUNCTION_PARAMVALUE (4051)
<u>Sleep</u>	ERR_CUSTOM_INDICATOR_ERROR (4055)
<u>StringFind</u>	ERR_STRING_PARAMETER_EXPECTED (4062)
<u>StringGetChar</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NOT_INITIALIZED_STRING (4008), ERR_ARRAY_INDEX_OUT_OF_RANGE (4002)
<u>StringLen</u>	ERR_STRING_PARAMETER_EXPECTED (4062)
<u>StringSetChar</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_INVALID_FUNCTION_PARAMVALUE (4051), ERR_NOT_INITIALIZED_STRING (4008), ERR_TOO_LONG_STRING (4011), ERR_ARRAY_INDEX_OUT_OF_RANGE (4002)
<u>StringSubstr</u>	ERR_STRING_PARAMETER_EXPECTED (4062), ERR_TOO_LONG_STRING (4011)
<u>StringTrimLeft</u>	ERR_STRING_PARAMETER_EXPECTED (4062)
<u>StringTrimRight</u>	ERR_STRING_PARAMETER_EXPECTED (4062)
<u>WindowIsVisible</u>	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059)
<u>WindowFind</u>	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NOT_INITIALIZED_STRING (4008)
<u>WindowHandle</u>	ERR_FUNC_NOT_ALLOWED_IN_TESTING (4059), ERR_STRING_PARAMETER_EXPECTED (4062), ERR_NOT_INITIALIZED_STRING (4008)
<u>WindowScreenShot</u>	ERR_WRONG_FILE_NAME (4101), ERR_INVALID_FUNCTION_PARAMVALUE (4051)

Остальные функции не меняют значение переменной last\_error ни при каких условиях.

AccountBalance, AccountCompany, AccountCredit, AccountCurrency, AccountEquity, AccountFreeMargin,  
AccountLeverage, AccountMargin, AccountName, AccountNumber, AccountProfit, AccountServer, Alert, CharToStr,

Comment, Day, DayOfWeek, DayOfYear, DoubleToStr, GetTickCount, HideTestIndicators, Hour, IndicatorCounted,  
IsConnected, IsDemo, IsDllsAllowed, IsExpertEnabled, IsLibrariesAllowed, IsOptimization, IsStopped, IsTesting,  
IsTradeAllowed, IsTradeContextBusy, IsVisualMode, MathAbs, MathArctan, MathCeil, MathCos, MathExp,  
MathFloor, MathLog, MathMax, MathMin, MathPow, MathRand, MathRound, MathSin, MathSrand, MathTan,  
Minute, Month, NormalizeDouble, ObjectsDeleteAll, ObjectsTotal, OrderSelect, OrdersHistoryTotal, Period, Print,  
RefreshRates, Seconds, SetLevelStyle, StringConcatenate, StrToTime, StrToDouble, Symbol, TerminalCompany,  
TerminalName, TerminalPath, TimeCurrent, TimeDay, TimeDayOfWeek, TimeDayOfYear, TimeHour, TimeLocal,  
TimeMinute, TimeMonth, TimeSeconds, TimeToStr, TimeYear, UninitializeReason, WindowBarsPerChart,  
WindowFirstVisibleBar, WindowPriceOnDropped, WindowTimeOnDropped, WindowsTotal, WindowOnDropped,  
WindowRedraw, WindowXOnDropped, WindowYOnDropped, Year

## **4 Предопределенные переменные**

Для каждой выполняющейся MQL4-программы поддерживается ряд предопределенных переменных, которые отражают состояние текущего ценового графика на момент запуска программы - эксперта, скрипта или пользовательского индикатора.

Библиотеки пользуются переменными вызвавшего их модуля.

Для безопасного и быстрого доступа к этим данным клиентский терминал обеспечивает локальные копии предопределенных переменных для каждой запущенной программы отдельно. Эти данные обновляются при каждом новом запуске прикрепленного эксперта или пользовательского индикатора автоматически, либо при помощи вызова функции [RefreshRates\(\)](#)

### **4.1 Ask**

**double Ask**

Последняя известная цена продажи (запрашиваемая цена) текущего инструмента. Для обновления необходимо использовать функцию [RefreshRates\(\)](#).

См. также [MarketInfo\(\)](#).

**Пример:**

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

### **4.2 Bars**

**int Bars**

Количество баров на текущем графике.

См. также [iBars\(\)](#).

**Пример:**

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

### **4.3 Bid**

**double Bid**

Последняя известная цена покупки (предложение на покупку) текущего инструмента. Для обновления необходимо использовать функцию [RefreshRates\(\)](#).

См. также [MarketInfo\(\)](#).

**Пример:**

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

### **4.4 Close**

**double Close[]**

Массив-таймсерия, содержащий цены закрытия каждого бара текущего графика.

Индексация элементов таймсерий производится задом наперед, от последнего к первому. Текущий бар, самый последний в массиве, имеет индекс 0. Самый старый бар, первый на графике, имеет индекс [Bars-1](#).

См. также [iClose\(\)](#).

**Пример:**

```
int handle = FileOpen("file.csv", FILE_CSV|FILE_WRITE, ";");
if(handle>0)
{
    // запись заголовков столбцов таблицы
    FileWrite(handle, "Time;Open;High;Low;Close;Volume");
    // запись данных
    for(int i=0; i<Bars; i++)
        FileWrite(handle, Time[i], Open[i], High[i], Low[i], Close[i],
Volume[i]);
    FileClose(handle);
}
```

### **4.5 Digits**

**int Digits**

Количество цифр после десятичной точки в цене текущего инструмента.

См. также [MarketInfo\(\)](#).

**Пример:**

```
Print(DoubleToStr(Close[0], Digits));
```

## **4.6 High**

**double High[]**

Массив-таймсерия, содержащий максимальные цены каждого бара текущего графика.

Индексация элементов таймсерий производится задом наперед, от последнего к первому. Текущий бар, самый последний в массиве, имеет индекс 0. Самый старый бар, первый на графике, имеет индекс Bars-1.

**См. также iHigh().**

**Пример:**

```
//---- maximums counting
i=Bars-KPeriod;
if(counted_bars>KPeriod) i=Bars-counted_bars-1;
while(i>=0)
{
double max=-1000000;
k=i+KPeriod-1;
while(k>=i)
{
price=High[k];
if(max<price) max=price;
k--;
}
HighesBuffer[i]=max;
i--;
}
//----
```

## **4.7 Low**

**double Low[]**

Массив-таймсерия, содержащий минимальные цены каждого бара текущего графика.

Индексация элементов таймсерий производится задом наперед, от последнего к первому. Текущий бар, самый последний в массиве, имеет индекс 0. Самый старый бар, первый на графике, имеет индекс Bars-1.

**См. также iLow().**

### Пример:

```
//---- minimums counting
i=Bars-KPeriod;
if(counted_bars>KPeriod) i=Bars-counted_bars-1;
while(i>=0)
{
    double min=1000000;
    k=i+KPeriod-1;
    while(k>=i)
    {
        price=Low[k];
        if(min>price) min=price;
        k--;
    }
    LowesBuffer[i]=min;
    i--;
}
```

## 4.8 Open

**double Open[]**

Массив-таймсерия, содержащий цены открытия каждого бара текущего графика.

Индексация элементов таймсерий производится задом наперед, от последнего к первому. Текущий бар, самый последний в массиве, имеет индекс 0. Самый старый бар, первый на графике, имеет индекс Bars-1.

См. также iOpen().

### Пример:

```
i = Bars - counted_bars - 1;
while(i>=0)
{
    double high = High[i];
    double low = Low[i];
    double open = Open[i];
    double close = Close[i];
    AccumulationBuffer[i] = (close-low) - (high-close);
    if(AccumulationBuffer[i] != 0)
    {
        double diff = high - low;
        if(0==diff)
            AccumulationBuffer[i] = 0;
        else
        {
            AccumulationBuffer[i] /= diff;
            AccumulationBuffer[i] *= Volume[i];
        }
    }
}
```

```

    }
    if(i<Bars-1) AccumulationBuffer[i] += AccumulationBuffer[i+1];
    i--;
}

```

## 4.9 Point

**double Point**

Размер пункта текущего инструмента в валюте котировки.

См. также [MarketInfo\(\)](#).

**Пример:**

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

## 4.10 Time

**datetime Time[]**

Массив-таймсерия, содержащий время открытия каждого бара текущего графика. Данные типа `datetime` представляют собой время в секундах, прошедшее с 00:00 1 января 1970 года.

Индексация элементов таймсерий производится задом наперед, от последнего к первому. Текущий бар, самый последний в массиве, имеет индекс 0. Самый старый бар, первый на графике, имеет индекс `Bars-1`.

См. также [iTime\(\)](#).

**Пример:**

```

for(i=Bars-2; i>=0; i--)
{
    if(High[i+1] > LastHigh) LastHigh = High[i+1];
    if(Low[i+1] < LastLow) LastLow = Low[i+1];
    //----
    if(TimeDay(Time[i]) != TimeDay(Time[i+1]))
    {
        P = (LastHigh + LastLow + Close[i+1])/3;
        R1 = P*2 - LastLow;
        S1 = P*2 - LastHigh;
        R2 = P + LastHigh - LastLow;
        S2 = P - (LastHigh - LastLow);
        R3 = P*2 + LastHigh - LastLow*2;
        S3 = P*2 - (LastHigh*2 - LastLow);
        LastLow = Open[i];
        LastHigh = Open[i];
    }
}

```

```

//----
PBuffer[i] = P;
S1Buffer[i] = S1;
R1Buffer[i] = R1;
S2Buffer[i] = S2;
R2Buffer[i] = R2;
S3Buffer[i] = S3;
R3Buffer[i] = R3;
}

```

## 4.11 Volume

**double Volume[]**

Массив-таймсерия, содержащий тиковые объемы каждого бара текущего графика.

Индексация элементов таймсерий производится задом наперед, от последнего к первому. Текущий бар, самый последний в массиве, имеет индекс 0. Самый старый бар, первый на графике, имеет индекс Bars-1.

**См. также iVolume().**

**Пример:**

```

if(i==0 && time0<i_time+periodseconds)
{
    d_volume += Volume[0];
    if(Low[0]<d_low)    d_low = Low[0];
    if(High[0]>d_high) d_high = High[0];
    d_close = Close[0];
}
last_fpos = FileTell(ExtHandle);
last_volume = Volume[i];
FileWriteInteger(ExtHandle, i_time, LONG_VALUE);
FileWriteDouble(ExtHandle, d_open, DOUBLE_VALUE);
FileWriteDouble(ExtHandle, d_low, DOUBLE_VALUE);
FileWriteDouble(ExtHandle, d_high, DOUBLE_VALUE);
FileWriteDouble(ExtHandle, d_close, DOUBLE_VALUE);
FileWriteDouble(ExtHandle, d_volume, DOUBLE_VALUE);

```

## 5 Информация о счете

### 5.1 AccountBalance

`double AccountBalance()`

Возвращает значение баланса активного счета (сумма денежных средств на счете).

**Пример:**

```
Print("Баланс счета = ", AccountBalance());
```

### 5.2 AccountCredit

`double AccountCredit()`

Возвращает значение кредита для активного счета.

**Пример:**

```
Print("Кредит счета ", AccountCredit());
```

### 5.3 AccountCompany

`string AccountCompany()`

Возвращает название брокерской компании, в которой зарегистрирован текущий счет.

**Пример:**

```
Print("Счет зарегистрирован в компании ", AccountCompany());
```

### 5.4 AccountCurrency

`string AccountCurrency()`

Возвращает наименование валюты для текущего счета.

**Пример:**

```
Print("Валюта счета", AccountCurrency());
```

### 5.5 AccountEquity

`double AccountEquity()`

Возвращает сумму собственных средств для текущего счета. Расчет equity зависит от настроек торгового сервера.

**Пример:**

```
Print("Средства счета = ",AccountEquity());
```

## **5.6 AccountFreeMargin**

**double AccountFreeMargin()**

Возвращает значение свободных средств, разрешенных для открытия позиций на текущем счете.

**Пример:**

```
Print("Свободная маржа счета = ",AccountFreeMargin());
```

## **5.7 AccountFreeMarginCheck**

**double AccountFreeMarginCheck ( string symbol, int cmd, double volume)**

Возвращает размер свободных средств, которые останутся после открытия указанной позиции по текущей цене на текущем счете. Если свободных средств не хватает, то будет сгенерирована ошибка 134 (ERR\_NOT\_ENOUGH\_MONEY).

**Параметры:**

- symbol** - Наименование финансового инструмента, с которым должна проводиться торговая операция.
- cmd** - Торговая операция. Может быть либо OP\_BUY, либо OP\_SELL.
- volume** - Количество лотов.

**Пример:**

```
if (AccountFreeMarginCheck (Symbol (), OP_BUY, Lots) <= 0 || GetLastError () == 134) return;
```

## **5.8 AccountFreeMarginMode**

**double AccountFreeMarginMode()**

Режим расчета свободных средств, разрешенных для открытия позиций на текущем счете.

Режим расчета может принимать следующие значения:

0 - при расчете не используются нереализованные прибыли и убытки;

1 - при расчете свободных средств используется как нереализованная прибыль, так и убыток по открытым позициям на текущем счете;

2 - при расчете используется только значение прибыли, текущий убыток по открытым

позициям не учитывается;

3 - при расчете используется только значение убытка, текущая прибыль по открытым позициям не учитывается.

**Пример:**

```
if (AccountFreeMarginMode() == 0)
    Print("Нереализованные прибыли/убытки не используются.");
```

### **5.9 AccountLeverage**

**int AccountLeverage()**

Возвращает значение плеча для текущего счета.

**Пример:**

```
Print("Счет #", AccountNumber(), " плечо ", AccountLeverage());
```

### **5.10 AccountMargin**

**double AccountMargin()**

Возвращает сумму залоговых средств, используемых для поддержания открытых позиций на текущем счете.

**Пример:**

```
Print("Сумма залоговых средств = ", AccountMargin());
```

### **5.11 AccountName**

**string AccountName()**

Возвращает имя пользователя текущего счета.

**Пример:**

```
Print("Имя = ", AccountName());
```

### **5.12 AccountNumber**

**int AccountNumber()**

Возвращает номер текущего счета.

**Пример:**

```
Print("Номер счета = ", AccountNumber());
```

### **5.13 AccountProfit**

**double AccountProfit()**

Возвращает значение прибыли для текущего счета в базовой валюте.

**Пример:**

```
Print("Прибыль ", AccountProfit());
```

### **5.14 AccountServer**

**string AccountServer()**

Возвращает имя активного сервера.

**Пример:**

```
Print("Адрес Сервера ", AccountServer());
```

### **5.15 AccountStopoutLevel**

**int AccountStopoutLevel()**

Возвращает значение уровня, по которому определяется состояние Stop Out.

**Пример:**

```
Print("StopOut level = ", AccountStopoutLevel());
```

### **5.16 AccountStopoutMode**

**int AccountStopoutMode()**

Возвращает режим расчета уровня Stop Out. Режим расчета может принимать следующие значения:

0 - расчет процентного соотношения залоговой маржи к средствам;

1 - сравнение уровня свободной маржи с абсолютным значением.

**Пример:**

```
int level=AccountStopoutLevel();
if(AccountStopoutMode()==0)
    Print("StopOut level = ", level, "%");
else
    Print("StopOut level = ", level, " ", AccountCurrency());
```

## **6 Операции с массивами**

Группа функций для работы с массивами.

Допускаются не более чем четырехмерные массивы. Индексация каждого измерения производится от 0 до **размер измерения-1**. В частном случае одномерного массива из 50 элементов обращение к первому элементу будет выглядеть как `array[0]`, к последнему элементу - `array[49]`.

При помощи данных функций (кроме тех функций, которые меняют качественные и количественные характеристики массива) могут обрабатываться и предопределенные массивы-таймсерии `Time[]`, `Open[]`, `High[]`, `Low[]`, `Close[]`, `Volume[]`

### **6.1 ArrayBsearch**

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

Возвращает индекс первого найденного элемента в первом измерении массива.

Если элемент с указанным значением в массиве отсутствует, функция вернет индекс ближайшего меньшего по значению из элементов, между которыми расположено искомое значение.

Функция не может применяться к массивами строк и таймсериям (исключение составляет таймсерия времени открытия бара).

Замечание: двоичный поиск обрабатывает только сортированные массивы. Для сортировки числового массива используется функция `ArraySort()`.

#### **Параметры:**

<b>array[]</b>	- Числовой массив для поиска.
<b>value</b>	- Значение для поиска.
<b>count</b>	- Количество элементов для поиска. По умолчанию, ищется в целом массиве.
<b>start</b>	- Начальный индекс для поиска. По умолчанию, поиск начинается с первого элемента.
<b>direction</b>	- Направление поиска. Может быть любой из следующих величин:

MODE\_ASCEND поиск в направлении возрастания,  
MODE\_DESCEND поиск в направлении убывания.

#### Пример:

```
datetime daytimes[];
int      shift=10,dayshift;
// Все Time[] серии времени отсортировано в направлении убывания
ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
if(Time[shift]>=daytimes[0]) dayshift=0;
else
{
    dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESCEND);
    if(Period()<PERIOD_D1) dayshift++;
}
Print(TimeToStr(Time[shift])," corresponds to ",dayshift," day bar opened
at ",
      TimeToStr(daytimes[dayshift]));
```

## 6.2 ArrayCopy

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

Копирует один массив в другой. Массивы должны иметь одинаковый тип. Массивы типа double[], int[], datetime[], color[], и bool[], можно копировать как массивы одного типа.

Возвращает количество скопированных элементов.

#### Параметры:

<b>dest[]</b>	- Массив-приемник.
<b>source[]</b>	- Массив-источник.
<b>start_dest</b>	- Начальный индекс для приемного массива. По умолчанию, стартовый индекс - 0.
<b>start_source</b>	- Начальный индекс для исходного массива. По умолчанию, стартовый индекс - 0.
<b>count</b>	- Количество элементов, которые нужно скопировать. По умолчанию, весь массив (WHOLE_ARRAY).

#### Пример:

```
double array1[][6];
double array2[10][6];
// array2 заполнен некоторыми данными
ArrayCopyRates(array1);
ArrayCopy(array2,array1,0,0,60);
```

```
// теперь array2 содержит первые 10 баров из истории (имеется в виду, что
первый бар - это бар с индексом [Bars-1])
ArrayCopy(array2,array1,0,Bars*6-60,60);
// теперь array2 содержит 10 последних баров из истории (имеется в виду,
что последний бар - это текущий бар, бар с индексом [0])
```

### 6.3 ArrayCopyRates

```
int void dest_array[], string symbol=NULL,
ArrayCopyRates( int timeframe=0)
```

Копирует в двухмерный массив, вида RateInfo[][6], данные баров текущего графика и возвращает количество скопированных баров, либо -1 в случае неудачи. Первое измерение содержит количество баров. Второе измерение имеет 6 элементов со значениями:

- 0 - время (time),
- 1 - цена открытия (open),
- 2 - наименьшая цена (low),
- 3 - наивысшая цена (high),
- 4 - цена закрытия (close),
- 5 - объём (volume).

Если копируются данные "чужого" инструмента и/или таймфрейма, то возможна ситуация отсутствия требуемых данных. В этом случае в переменную last\_error будет помещена ошибка ERR\_HISTORY\_WILL\_UPDATED (4066 - запрошенные исторические данные в состоянии обновления) и необходимо через некоторое время повторить попытку копирования.

Замечания: обычно массив используется, чтобы передать данные в DLL функцию.

Реального распределения памяти под массив данных и копирования не происходит. При обращении к такому массиву производится перенаправление доступа.

#### Параметры:

```
dest_array[] - Ссылка на двумерный массив.
symbol - Наименование инструмента (символ валютной пары).
timeframe - Период. Может быть любым значением из перечисленных периодов.
```

#### Пример:

```
double array1[][6];
```

```
ArrayCopyRates(array1,"EURUSD", PERIOD_H1);
Print("Текущий бар ",TimeToStr(array1[0][0]),"цена открытия ",
array1[0][1]);
```

## 6.4 ArrayCopySeries

```
int void array[], int series_index,
ArrayCopySeries( string symbol=NULL, int timeframe=0)
```

Копирует массив-таймсерию в пользовательский массив и возвращает количество скопированных элементов.

Реального распределения памяти под массив данных и копирования не происходит. При обращении к такому массиву производится перенаправление доступа. Исключения составляют массивы, назначенные в качестве индексных в пользовательских индикаторах. В этом случае производится реальное копирование данных.

Если копируются данные "чужого" инструмента и/или таймфрейма, то возможна ситуация отсутствия требуемых данных. В этом случае в переменную last\_error будет помещена ошибка ERR\_HISTORY\_WILL\_UPDATED (4066 - запрошенные исторические данные в состоянии обновления) и необходимо через некоторое время повторить попытку копирования.

Замечание: если series\_index - MODE\_TIME, то передаваемый в функцию массив должен иметь тип datetime[].

### Параметры:

<b>array[]</b>	- Ссылка на одномерный числовой массив.
<b>series_index</b>	- Идентификатор массива-таймсерии. Должен быть одним из перечисленных идентификаторов <u>таймсерий</u> .
<b>symbol</b>	- Наименование инструмента (символ валютной пары).
<b>timeframe</b>	- Период графика (таймфрейм). Может быть любым значением из <u>перечисленных периодов</u> .

### Пример:

```
datetime daytimes[];
int shift=10,dayshift,error;
//---- массив Time[] отсортирован в порядке убывания
ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
```

```

error=GetLastError();
if(error==4066)
{
    //---- делаем еще 2 попытки чтения
    for(int i=0;i<2; i++)
    {
        Sleep(5000);
        ArrayCopySeries(daytimes,MODE_TIME,Symbol(),PERIOD_D1);
        //---- проверим время текущего дневного бара
        datetime last_day=daytimes[0];
        if(Year()==TimeYear(last_day) && Month()==TimeMonth(last_day) &&
Day()==TimeDay(last_day)) break;
    }
}
if(Time[shift]>=daytimes[0]) dayshift=0;
else
{
    dayshift=ArrayBsearch(daytimes,Time[shift],WHOLE_ARRAY,0,MODE_DESCEND);
    if(Period(<PERIOD_D1) dayshift++;
}
Print(TimeToStr(Time[shift])," corresponds to ",dayshift," day bar opened
at ", TimeToStr(daytimes[dayshift]));

```

## 6.5 ArrayDimension

**int ArrayDimension(object array[])**

Возвращает ранг многомерного массива.

**Параметры:**

**array[]** - Массив, для которого будет возвращен ранг.

**Пример:**

```

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

```

## 6.6 ArrayGetAsSeries

**bool ArrayGetAsSeries(object array[])**

Возвращается TRUE, если массив организован как таймсерия (элементы массива индексируются от последнего к первому), иначе возвращается FALSE.

**Параметры:**

**array[]** - Проверяемый массив.

**Пример:**

```

if(ArrayGetAsSeries(array1)==true)

```

```
Print("array1 индексирован как таймсерия");
else
Print("array1 индексирован обычно (слева направо)");
```

## 6.7 ArrayInitialize

**int ArrayInitialize(void array[], double value)**

Устанавливает все элементы числового массива в одну величину. Возвращает количество инициализированных элементов.

Замечание: не рекомендуется инициализировать индексные буферы в функции *init()* пользовательских индикаторов, так как они инициализируются автоматически "пустым значением" при распределении и перераспределении буферов.

### Параметры:

**array[]** - Числовой массив, который нужно инициализировать.

**value** - Новая величина, которая нужно установить.

### Пример:

```
//---- инициализация всех элементов массива значением 2.1
double myarray[10];
ArrayInitialize(myarray, 2.1);
```

## 6.8 ArrayIsSeries

**bool ArrayIsSeries(object array[])**

Возвращается TRUE, если проверяемый массив является массивом-таймсерией (Time[], Open[], Close[], High[], Low[] или Volume[]), иначе возвращается FALSE.

### Параметры:

**array[]** - Проверяемый массив.

### Пример:

```
if(ArrayIsSeries(array1)==false)
ArrayInitialize(array1, 0);
else
{
Print("Таймсерия не может быть инициализирована!");
return(-1);
}
```

## 6.9 ArrayMaximum

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

Поиск элемента с максимальным значением. Функция возвращает позицию максимального элемента в массиве.

### Параметры:

- array[]** - Числовой массив, в котором производится поиск.
- count** - Количество элементов для поиска.
- start** - Начальный индекс для поиска.

### Пример:

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

## 6.10 ArrayMinimum

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

Поиск элемента с минимальным значением. Функция возвращает позицию минимального элемента в массиве.

### Параметры:

- array[]** - Числовой массив, в котором производится поиск.
- count** - Количество элементов для поиска.
- start** - Начальный индекс для поиска.

### Пример:

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

## 6.11 ArrayRange

```
int ArrayRange(object array[], int range_index)
```

Возвращает число элементов в указанном измерении массива. Поскольку индексы начинаются с нуля, размер измерения на 1 больше, чем самый большой индекс.

### Параметры:

**array[]** - Проверяемый массив  
**range\_index** - Индекс измерения.

**Пример:**

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

## 6.12 ArrayResize

**int ArrayResize(void array[], int new\_size)**

Устанавливает новый размер в первом измерении массива. При успешном выполнении функция возвращает количество всех элементов, содержащихся в массиве после изменения размера, в противном случае возвращает -1, и массив не меняет размеры.

Замечание: массив, объявленный на локальном уровне в какой-либо функции, у которого был изменен размер, останется неизменным после завершения работы функции. При повторном вызове функции такой массив будет иметь размер, отличный от объявленного.

**Параметры:**

**array[]** - Массив для изменения размеров.  
**new\_size** - Новый размер для первого измерения.

**Пример:**

```
double array1[][4];  
int    element_count=ArrayResize(array1, 20);  
// новый размер - 80 элементов
```

## 6.13 ArraySetAsSeries

**bool ArraySetAsSeries(void array[], bool set)**

Устанавливает направление индексирования в массиве. Значение параметра *set* TRUE устанавливает направление индексирования в обратном порядке, то есть, последний элемент имеет нулевой индекс. Значение FALSE устанавливает нормальное направление индексирования. Функция возвращает предыдущее состояние.

**Параметры:**

**array[]** - Числовой массив для установки.  
**set** - Направление индексирования массива.

**Пример:**

```

double macd_buffer[300];
double signal_buffer[300];
int    i, limit=ArraySize(macd_buffer);
ArraySetAsSeries(macd_buffer, true);

for(i=0; i<limit; i++)
    macd_buffer[i]=iMA(NULL, 0, 12, 0, MODE_EMA, PRICE_CLOSE, i) -
iMA(NULL, 0, 26, 0, MODE_EMA, PRICE_CLOSE, i);

for(i=0; i<limit; i++)
    signal_buffer[i]=iMAOnArray(macd_buffer, limit, 9, 0, MODE_SMA, i);

```

## 6.14 ArraySize

**int ArraySize(object array[])**

Возвращает количество элементов массива. Для одномерного массива значение, возвращаемое функцией ArraySize, равно значению ArrayRange(array,0).

**Параметры:**

**array[]** - Массив любого типа.

**Пример:**

```

int count=ArraySize(array1);
for(int i=0; i<count; i++)
{
    // некоторые вычисления.
}

```

## 6.15 ArraySort

**int void array[], int count=WHOLE\_ARRAY, int start=0, ArraySort( int sort\_dir=MODE\_ASCEND)**

Сортировка числовых массивов по первому измерению. Массивы-таймсерии не могут быть отсортированы.

**Параметры:**

**array[]** - Числовой массив для сортировки.

**count** - Количество элементов для сортировки.

**start** - Начальный индекс.

**sort\_dir** - Направление сортировки массива. Это может быть любым из следующего величин  
MODE\_ASCEND - сортировка в порядке возрастания

MODE\_DESCEND - сортировка в порядке убывания.

**Пример:**

```
double num_array[5]={4,1,6,3,9};  
// массив содержит величины 4,1,6,3,9  
ArraySort(num_array);  
// теперь массив отсортирован 1,3,4,6,9  
ArraySort(num_array,WHOLE_ARRAY,0,MODE_DESCEND);  
// теперь массив отсортирован 9,6,4,3,1
```

## **7 Проверка состояния**

Группа функций, позволяющих определить текущее состояние клиентского терминала, в том числе состояние окружения выполняемой MQL4-программы

### **7.1 GetLastError**

**int GetLastError()**

Функция возвращает код последней ошибки, после чего значение специальной переменной last\_error, в которой хранится код последней ошибки обнуляется. Так что последующий вызов GetLastError() вернет значение 0.

**Пример:**

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

### **7.2 IsConnected**

**bool IsConnected()**

Возвращает состояние главного соединения клиентского терминала с сервером, по которому производится подкачка данных. TRUE - связь с сервером установлена, FALSE - связь с сервером отсутствует или прервана.

**Пример:**

```
if(!IsConnected())
{
    Print("Связь отсутствует!");
    return(0);
}
// Тело скрипта, нуждающегося в открытом подключении
// ...
```

### **7.3 IsDemo**

**bool IsDemo()**

Возвращается TRUE, если программа работает на демонстрационном счете, в противном случае возвращает FALSE.

### Пример:

```
if(IsDemo()) Print("Я работаю на демонстрационном счете");
else Print("Я работаю на реальном счете");
```

## 7.4 IsDllsAllowed

**bool IsDllsAllowed()**

Возвращает TRUE, если DLL вызов функции разрешены для эксперта, иначе возвращает FALSE.

См. также [IsLibrariesAllowed\(\)](#), [IsTradeAllowed\(\)](#).

### Пример:

```
#import "user32.dll"
    int      MessageBoxA(int hWnd, string szText, string szCaption,int
nType);
    ...
    ...
    if(IsDllsAllowed()==false)
    {
        Print("Вызов из библиотек (DLL) невозможен. Эксперт не может
выполняться.");
        return(0);
    }
    // Тело Эксперта, вызывающее внешние функции DLL
    MessageBoxA(0,"an message","Message",MB_OK);
```

## 7.5 IsExpertEnabled

Страница не читаема

## 7.6 IsLibrariesAllowed

**bool IsLibrariesAllowed()**

Возвращает TRUE, если эксперт может назвать библиотечную функцию, иначе возвращает FALSE.

См. также [IsDllsAllowed\(\)](#), [IsTradeAllowed\(\)](#).

### Пример:

```
#import "somelibrary.ex4"
    int somefunc();
    ...
    ...
    if(IsLibrariesAllowed()==false)
    {
        Print("Библиотечные вызовы запрещены.");
        return(0);
    }
```

```
// Тело Эксперта, вызывающее внешние функции DLL
somefunc();
```

### **7.7 IsOptimization**

**bool IsOptimization()**

Возвращается TRUE, если эксперт работает в режиме оптимизации тестирования, иначе возвращает FALSE.

**Пример:**

```
if(IsOptimization()) return(0);
```

### **7.8 IsStopped**

**bool IsStopped()**

Возвращается TRUE, если программа (эксперт или скрипт) получила команду на завершение своей работы, иначе возвращает FALSE. Программа может работать еще 2.5 секунды прежде, чем клиентский терминал принудительно завершит ее выполнение.

**Пример:**

```
while(expr!=false)
{
    if(IsStopped()==true) return(0);
    // цикл с длительным временем исполнения
    // ...
}
```

### **7.9 IsTesting**

**bool IsTesting()**

Возвращается TRUE, если эксперт работает в режиме тестирования, иначе возвращает FALSE.

**Пример:**

```
if(IsTesting()) Print("Тестирование эксперта");
```

### **7.10 IsTradeAllowed**

**bool IsTradeAllowed()**

Возвращается TRUE, если эксперту разрешено торговать и поток для выполнения торговых операций свободен, иначе возвращает FALSE.

**См. также** IsDllsAllowed(), IsLibrariesAllowed(), IsTradeContextBusy().

**Пример:**

```
if(IsTradeAllowed()) Print("Торговля разрешена");
```

### **7.11 IsTradeContextBusy**

**bool IsTradeContextBusy()**

Возвращается TRUE, если поток для выполнения торговых операций занят, иначе возвращает FALSE.

См. также [IsTradeAllowed\(\)](#).

**Пример:**

```
if(IsTradeContextBusy()) Print("Торговый поток занят. Подождите");
```

### **7.12 IsVisualMode**

**bool IsVisualMode()**

Возвращается TRUE, если эксперт тестируется в режиме визуализации, иначе возвращает FALSE.

**Пример:**

```
if(IsVisualMode()) Comment("Визуализация включена");
```

### **7.13 UninitializeReason**

**int UninitializeReason()**

Возвращает код причины завершения экспертов, пользовательских индикаторов и скриптов. Возвращаемые значения могут быть одним из [кодов деинициализации](#). Эту функцию можно также вызывать в функции `init()` для анализа причин деинициализации предыдущего запуска.

**Пример:**

```
// пример
int deinit()
{
    switch(UninitializeReason())
    {
        case REASON_CHARTCLOSE:
        case REASON_REMOVE:      CleanUp(); break;      // очистка и
освобождение ресурсов.
        case REASON_RECOMPILE:
        case REASON_CHARTCHANGE:
        case REASON_PARAMETERS:
```

```
        case REASON_ACCOUNT:      StoreData(); break; // подготовка к
рестарту.
    }
    //...
}
```

## **8 Клиентский терминал**

Функции, возвращающие информацию о клиентском терминале

### **8.1 TerminalCompany**

**string TerminalCompany()**

Возвращает наименование компании-владельца клиентского терминала.

**Пример:**

```
Print("Компания ", TerminalCompany());
```

### **8.2 TerminalName**

**string TerminalName()**

Возвращает имя клиентского терминала.

**Пример:**

```
Print("Работает терминал ", TerminalName());
```

### **8.3 TerminalPath**

**string TerminalPath()**

Возвращает директорию, из которого запущен клиентский терминал.

**Пример:**

```
Print("Рабочий директорию ", TerminalPath());
```

## 9 Общие функции

Функции общего назначения, которые не вошли ни в одну из специализированных групп

### 9.1 *Alert*

`void Alert(...)`

Отображает диалоговое окно, содержащие пользовательские данные. Параметры могут быть любого типа. Количество параметров не может превышать 64.

Массивы нельзя передавать в функцию `Alert()`. Массивы должны выводиться поэлементно.

Данные типа `double` выводятся с 4 десятичными цифрами после точки. Для вывода чисел с большей точностью используйте функцию `DoubleToStr()`.

Данные типы `bool`, `datetime` и `color` будут выведены как числа.

Чтобы вывести данные типа `datetime` в виде строки, необходимо использовать функцию `TimeToStr()`.

Для разделения выводимой информации на несколько строк можно использовать символ перевода строки `"\n"` либо `"\r\n"`.

См. также функции `Comment()` и `Print()`.

#### Параметры:

... - Любые значения, разделенные запятыми.

#### Пример:

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

### 9.2 *Comment*

`void Comment(...)`

Функция выводит комментарий, определенный пользователем, в левый верхний угол графика. Параметры могут иметь любой тип. Количество параметров не может превышать 64.

Массивы нельзя передавать в функцию `Comment()`. Массивы должны печататься

поэлементно.

Данные типа `double` выводятся с 4 десятичными цифрами после точки. Для вывода чисел с большей точностью необходимо использовать функцию `DoubleToStr()`.

Типы `bool`, `datetime` и `color` будут напечатаны как числа.

Чтобы вывести данные типа `datetime` в виде строки используйте функцию `TimeToStr()`.

Для разделения выводимой информации на несколько строк можно использовать символ перевода строки `"\n"` либо `"\r\n"`.

См. также функции `Alert()` и `Print()`.

#### Параметры:

... - Любые значения, разделенные запятыми.

#### Пример:

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

### 9.3 GetTickCount

`int GetTickCount()`

Функция `GetTickCount()` возвращает количество миллисекунд, прошедших с момента старта системы. Счетчик ограничен разрешающей способностью системного таймера. Так как время хранится как беззнаковое целое, то он переполняется каждые 49.7 дней.

#### Пример:

```
int start=GetTickCount();
// некие серьезные вычисления...
Print("Время вычисления ", GetTickCount()-start, " миллисекунд.");
```

### 9.4 MarketInfo

`double MarketInfo(string symbol, int type)`

Возвращает различную информацию о финансовых инструментах, перечисленных в окне "Обзор рынка". Часть информации о текущем финансовом инструменте хранится в предопределенных переменных.

#### Параметры:

**symbol** - Символ инструмента.

**type** - Идентификатор запроса, определяющий тип возвращаемой информации. Может быть любым из значений идентификаторов запроса.

**Пример:**

```
double bid    =MarketInfo("EURUSD",MODE_BID);
double ask    =MarketInfo("EURUSD",MODE_ASK);
double point  =MarketInfo("EURUSD",MODE_POINT);
int    digits=MarketInfo("EURUSD",MODE_DIGITS);
int    spread=MarketInfo("EURUSD",MODE_SPREAD);
```

## 9.5 MessageBox

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

Функция MessageBox создает и отображает окно сообщений, а также управляет им. Окно сообщений содержит определенное приложением сообщение и заголовок, любую комбинацию predefined значков и командных кнопок. Если функция успешно выполняется, возвращаемое значение - одно из значений кодов возврата MessageBox(). Функцию нельзя вызывать из пользовательских индикаторов, так как индикаторы выполняются в интерфейсном потоке и не должны его тормозить.

**Параметры:**

**text** - Текст, содержащий сообщение для отображения.

**caption** - Необязательный текст для отображения в заголовке окна сообщения. Если этот параметр пустой, в заголовке окна будет отображено название эксперта.

**flags** - Необязательные флаги, определяющие вид и поведение диалогового окна. Флаги могут быть комбинацией флагов из следующих групп флагов.

**Пример:**

```
#include <WinUser32.mqh>

if(ObjectCreate("text_object", OBJ_TEXT, 0, D'2004.02.20 12:30',
1.0045)==false)
{
    int ret=MessageBox("Функция ObjectCreate() вернула ошибку
"+GetLastError()+"\nПродолжить?", "Question", MB_YESNO|MB_ICONQUESTION);
    if(ret==IDNO) return(false);
}
```

```
// продолжение
```

## 9.6 PlaySound

```
void PlaySound(string filename)
```

Функция воспроизводит звуковой файл. Файл должен быть расположен в каталоге *каталог\_терминала\sounds* или его подкаталоге.

### Параметры:

**filename** - путь к звуковому файлу.

### Пример:

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

## 9.7 Print

```
void Print(...)
```

Печатает некоторое сообщение в журнал экспертов. Параметры могут иметь любой тип. Количество параметров не может превышать 64.

Массивы нельзя передать в функцию Print(). Массивы должны быть напечатаны поэлементно.

Данные типа double выводятся с 4 десятичными цифрами после точки. Чтобы получить большую точность, следует использовать функцию DoubleToStr().

Данные типов bool, datetime и color будут напечатаны в виде чисел.

Чтобы печатать значения datetime как строку с датой, следует использовать функцию TimeToStr().

См. также функции Alert() и Comment().

### Параметры:

**...** - Любые значения, разделенные запятыми.

### Пример:

```
Print("Свободная маржа счета ", AccountFreeMargin());  
Print("Текущее время ", TimeToStr(TimeCurrent()));  
double pi=3.141592653589793;  
Print("Число PI ", DoubleToStr(pi,8));  
// Выход: число PI 3.14159265  
// Печать массива  
for(int i=0;i<10;i++)
```

```
Print(Close[i]);
```

## 9.8 **SendFTP**

**bool SendFTP(string filename, string ftp\_path=NULL)**

Посылает файл по адресу, указанному в окне настроек на закладке "Публикация". В случае неудачи возвращает FALSE.

Функция не работает в режиме тестирования. Из пользовательских индикаторов также нельзя вызывать эту функцию.

Отсылаемый файл должен находиться в папке *каталог\_терминала\experts\files* или ее подпапках.

Отсылка не производится, если в настройках не указан адрес FTP и/или пароль доступа.

### Параметры:

**filename** - Имя отсылаемого файла.

**ftp\_path** - Каталог FTP. Если каталог не указан, то используется каталог, описанный в настройках.

### Пример:

```
int lasterror=0;
if(!SendFTP("report.txt"))
    lasterror=GetLastError();
```

## 9.9 **SendMail**

**void SendMail(string subject, string some\_text)**

Посылает электронное письмо по адресу, указанному в окне настроек на закладке "Почта".

Отсылка может быть запрещена в настройках, также может быть не указан адрес электронной почты. Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

### Параметры:

**subject** - Заголовок письма.

**some\_text** - Тело письма.

### Пример:

```
double lastclose=Close[0];
if(lastclose<my_signal)
```

```
SendMail("из Вашего эксперта", "Цена изменилась  
"+DoubleToStr(lastclose,Digits));
```

## 9.10 Sleep

**void Sleep(int milliseconds)**

Функция задерживает выполнение текущего эксперта или скрипта на определенный интервал.

Функцию Sleep() нельзя вызывать из пользовательских индикаторов, так как индикаторы выполняются в интерфейсном потоке и не должны его тормозить.

В функцию встроена проверка состояния флага остановки эксперта каждую 0.1 секунды.

**Параметры:**

**milliseconds** - Интервал задержки в миллисекундах.

**Пример:**

```
//---- wait for 10 seconds  
Sleep(10000);
```

## 10 Преобразования данных

Группа функций, обеспечивающих преобразование данных из одного формата в данные другого формата.

Особо следует отметить функцию `NormalizeDouble()`, которая обеспечивает требуемую точность представления цены. В торговых операциях нельзя использовать ненормализованные цены, чья точность превышает требуемую торговым сервером хотя бы на один знак

### 10.1 CharToStr

`string CharToStr(int char_code)`

Преобразование кода символа в односимвольную строку.

**Параметры:**

**char\_code** - ASCII-код символа.

**Пример:**

```
string str="WORLD"+CharToStr(44); // 44 - код для 'D'  
// результирующая строка будет WORLD
```

### 10.2 DoubleToStr

`string DoubleToStr(double value, int digits)`

Преобразование числового значения в текстовую строку, содержащую символьное представление числа в указанном формате точности.

**Параметры:**

**value** - Величина с плавающей точкой.

**digits** - Формат точности, число цифр после десятичной точки (0-8).

**Пример:**

```
string value=DoubleToStr(1.28473418, 5);  
// содержимое строки value - "1.28473"
```

### 10.3 NormalizeDouble

`double NormalizeDouble(double value, int digits)`

Округление числа с плавающей запятой до указанной точности.

Рассчитываемые значения StopLoss, TakeProfit, а также значения цены открытия отложенных ордеров должны быть нормализованы с точностью, значение которой хранится в предопределенной переменной Digits.

**Параметры:**

**value** - Величина с плавающей точкой.

**digits** - Формат точности, число цифр после десятичной точки (0-8).

**Пример:**

```
double var1=0.123456789;  
Print(DoubleToStr(NormalizeDouble(var1,5),8));  
// вывод: 0.12346000
```

## **10.4 StrToDouble**

**double StrToDouble(string value)**

Преобразование строки, содержащей символьное представление числа, в число типа double (формат двойной точности с плавающей точкой).

**Параметры:**

**value** - Строка, содержащая символьное представление числа.

**Пример:**

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

## **10.5 StrToInteger**

**int StrToInteger(string value)**

Преобразование строки, содержащей символьное представление числа, в число типа int (целое).

**Параметры:**

**value** - Строка, содержащая число.

**Пример:**

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

## **10.6 StrToTime**

**datetime StrToTime(string value)**

Преобразование строки, содержащей время и/или дату в формате "yyyy.mm.dd [hh:mi]", в число типа `datetime` (количество секунд, прошедших с 01.01.1970).

**Параметры:**

**value** - Строка в формате "yyyy.mm.dd hh:mi".

**Пример:**

```
datetime var1, var2, var3;
var1=StrToTime("2003.8.12 17:35");
var2=StrToTime("17:35"); // возврат текущей даты с указанным
временем
var3=StrToTime("2003.8.12"); // возврат даты с полуночным временем
"00:00"
```

### **10.7 TimeToStr**

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

Преобразование значения, содержащего время в секундах, прошедшее с 01.01.1970, в строку формата "yyyy.mm.dd hh:mi".

**Параметры:**

**value** - Время в секундах от 00:00 1 января 1970.

**mode** - Дополнительный режим вывода данных. Может быть одним или комбинированным флагом:

TIME\_DATE получает результат в форме "yyyy.mm.dd",

TIME\_MINUTES получает результат в форме "hh:mi",

TIME\_SECONDS получает результат в форме "hh:mi:ss".

**Пример:**

```
string var1=TimeToStr(TimeCurrent(), TIME_DATE|TIME_SECONDS);
```

## 11 Пользовательские индикаторы

Группа функций, используемых при оформлении пользовательских индикаторов.

Данные функции нельзя использовать при написании советников и скриптов.

### 11.1 IndicatorBuffers

**void IndicatorBuffers(int count)**

Распределяет память для буферов, используемых для вычислений пользовательского индикатора. Количество буферов не может превышать 8 и быть менее значения, указанного в свойстве `indicator_buffers`. Если пользовательский индикатор требует дополнительных буферов для счета, следует использовать эту функцию для указания общего числа буферов.

**Параметры:**

**count** - Количество расчетных буферов. от `indicator_buffers` до 8 буферов.

**Пример:**

```
#property indicator_separate_window
#property indicator_buffers 1
#property indicator_color1 Silver
//---- параметры индикатора
extern int FastEMA=12;
extern int SlowEMA=26;
extern int SignalSMA=9;
//---- буферы индикатора
double ind_buffer1[];
double ind_buffer2[];
double ind_buffer3[];
//+-----+
//| специальная функция инициализации индикатора |
//+-----+
int init()
{
//---- 2 дополнительных буфера, используемых для расчета.
IndicatorBuffers(3);
//---- параметры рисования
SetIndexStyle(0, DRAW_HISTOGRAM, STYLE_SOLID, 3);
SetIndexDrawBegin(0, SignalSMA);
IndicatorDigits(MarketInfo(Symbol(), MODE_DIGITS)+2);
//---- 3 распределенных буфера индикатора
SetIndexBuffer(0, ind_buffer1);
SetIndexBuffer(1, ind_buffer2);
SetIndexBuffer(2, ind_buffer3);
//---- маркировка именами для DataWindow и подокна (subwindow) индикатора
IndicatorShortName("OsMA("+FastEMA+", "+SlowEMA+", "+SignalSMA+")");
```

```
//---- инициализация завершена
return(0);
}
```

## 11.2 IndicatorCounted

**int IndicatorCounted()**

Функция возвращает количество баров, не измененных после последнего вызова индикатора. Большинство подсчитанных баров не нуждается в пересчете. Функция используется для оптимизации вычислений.

Замечание: самый последний бар не считается посчитанным, и в большинстве случаев необходимо пересчитывать только его. Однако бывают пограничные случаи, когда вызов пользовательского индикатора производится из эксперта на первом тике нового бара. Возможна ситуация, что последний тик предыдущего бара не обработан (по той причине, что в момент прихода этого последнего тика обрабатывался предпоследний тик), и пользовательский индикатор не был вызван и поэтому не был рассчитан. Чтобы избежать в такой ситуации ошибок расчета индикатора, функция IndicatorCounted() возвращает реально посчитанное количество баров минус один.

**Пример:**

```
int start()
{
    int limit;
    int counted_bars=IndicatorCounted();
    //---- последний посчитанный бар будет пересчитан
    if(counted_bars>0) counted_bars--;
    limit=Bars-counted_bars;
    //---- основной цикл
    for(int i=0; i<limit; i++)
    {
        //---- ma_shift присваивается 0, потому что SetIndexShift,
        вызываемый выше
        ExtBlueBuffer[i]=iMA(NULL,0,JawsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
        ExtRedBuffer[i]=iMA(NULL,0,TeethPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
        ExtLimeBuffer[i]=iMA(NULL,0,LipsPeriod,0,MODE_SMMA,PRICE_MEDIAN,i);
    }
    //----
    return(0);
}
```

## 11.3 IndicatorDigits

**void IndicatorDigits(int digits)**

Установка формата точности (количество знаков после десятичной точки) для визуализации значений индикатора. По умолчанию используется точность цены финансового инструмента, к графику которого присоединен индикатор.

#### Параметры:

**digits** - Формат точности, число цифр после десятичной точки.

#### Пример:

```
int init()
{
//---- 2 дополнительных буфера используются для подсчета.
    IndicatorBuffers(3);
//---- настройка параметров отрисовки
    SetIndexStyle(0, DRAW_HISTOGRAM, STYLE_SOLID, 3);
    SetIndexDrawBegin(0, SignalSMA);
    IndicatorDigits(Digits+2);
//---- 3 распределенных буферов индикатора
    SetIndexBuffer(0, ind_buffer1);
    SetIndexBuffer(1, ind_buffer2);
    SetIndexBuffer(2, ind_buffer3);
//---- "короткое имя" для DataWindow и подокна индикатора
    IndicatorShortName("OsMA("+FastEMA+", "+SlowEMA+", "+SignalSMA+)");
//---- инициализация сделана
    return(0);
}
```

### 11.4 IndicatorShortName

**void IndicatorShortName(string name)**

Установка "короткого" имени пользовательского индикатора для отображения в подокне индикатора и в окне DataWindow.

#### Параметры:

**name** - Новое короткое имя.

#### Пример:

```
int init()
{
//---- 2 дополнительных буфера используются для подсчета.
    IndicatorBuffers(3);
//---- настройка параметров отрисовки
    SetIndexStyle(0, DRAW_HISTOGRAM, STYLE_SOLID, 3);
    SetIndexDrawBegin(0, SignalSMA);
    IndicatorDigits(Digits+2);
//---- 3 распределенных буферов индикатора
    SetIndexBuffer(0, ind_buffer1);
    SetIndexBuffer(1, ind_buffer2);
    SetIndexBuffer(2, ind_buffer3);
//---- "короткое имя" для DataWindow и подокна индикатора
```

```

IndicatorShortName("OsMA("+FastEMA+", "+SlowEMA+", "+SignalSMA+")");
//---- инициализация сделана
return(0);
}

```

## 11.5 SetIndexArrow

**void SetIndexArrow(int index, int code)**

Назначение значка для линии индикаторов, имеющей стиль DRAW\_ARROW.

Нельзя использовать коды стрелок вне диапазона 33-255.

**Параметры:**

- index** - Порядковый номер линии. Должен быть от 0 до 7.
- code** - Код символа из шрифта Wingdings или одним из предопределенных стрелок.

**Пример:**

```

int init()
{
//---- 2 распределенных буфера индикатора
SetIndexBuffer(0,ExtUpperBuffer);
SetIndexBuffer(1,ExtLowerBuffer);
//---- настройка параметров отрисовки
SetIndexStyle(0,DRAW_ARROW);
SetIndexArrow(0,217);
SetIndexStyle(1,DRAW_ARROW);
SetIndexArrow(1,218);
//---- отображение в DataWindow
SetIndexLabel(0,"Fractal Up");
SetIndexLabel(1,"Fractal Down");
//---- инициализация сделана
return(0);
}

```

## 11.6 SetIndexBuffer

**bool SetIndexBuffer(int index, double array[])**

Связывает переменную-массив, объявленный на глобальном уровне, с предопределенным буфером пользовательского индикатора. Количество буферов, необходимых для расчета индикатора, задается с помощью функции IndicatorBuffers() и не может быть больше 8. В случае успешного связывания возвращается TRUE, иначе FALSE. Чтобы получить расширенные сведения об ошибке, следует вызвать функцию GetLastError().

**Параметры:**

**index** - Порядковый номер линии. Должен быть от 0 до 7.

**array[]** - Ссылка на массив, который будет связан с расчетным буфером.

**Пример:**

```
double ExtBufferSilver[];
int init()
{
    SetIndexBuffer(0, ExtBufferSilver); // буфер для первой линии
    // ...
}
```

### **11.7 SetIndexDrawBegin**

**void SetIndexDrawBegin(int index, int begin)**

Установка порядкового номера бара от начала данных, с которого должна начинаться отрисовка указанной линии индикатора. Отрисовка индикатора производится слева направо. Значения индикаторного массива, находящиеся левее указанного бара, не будут рисоваться на графике и отображаться в окне DataWindow. По умолчанию устанавливается значение 0.

**Параметры:**

**index** - Порядковый номер линии. Должен быть от 0 до 7.

**begin** - Номер позиции начала отрисовки линии индикатора.

**Пример:**

```
int init()
{
    //---- 2 дополнительных буфера используются для подсчета.
    IndicatorBuffers(3);
    //---- настройка параметров рисунка
    SetIndexStyle(0, DRAW_HISTOGRAM, STYLE_SOLID, 3);
    SetIndexDrawBegin(0, SignalSMA);
    IndicatorDigits(Digits+2);
    //---- 3 распределенных буферов индикатора
    SetIndexBuffer(0, ind_buffer1);
    SetIndexBuffer(1, ind_buffer2);
    SetIndexBuffer(2, ind_buffer3);
    //---- "короткое имя" для DataWindow и подокна индикатора
    IndicatorShortName("OsMA("+FastEMA+", "+SlowEMA+", "+SignalSMA+)");
    //---- инициализация сделана
    return(0);
}
```

## 11.8 SetIndexEmptyValue

`void SetIndexEmptyValue(int index, double value)`

Устанавливает значение пустой величины для линии индикатора. Пустые значения не рисуются и не показываются в DataWindow. По умолчанию значение пустой величины - EMPTY\_VALUE.

### Параметры:

**index** - Порядковый номер линии. Должен быть от 0 до 7.

**value** - Новое "пустое" значение.

### Пример:

```
int init()
{
//---- 2 распределенных буфера индикатора
    SetIndexBuffer(0,ExtUppperBuffer);
    SetIndexBuffer(1,ExtLowerBuffer);
//---- настройка параметров отрисовки
    SetIndexStyle(0,DRAW_ARROW);
    SetIndexArrow(0,217);
    SetIndexStyle(1,DRAW_ARROW);
    SetIndexArrow(1,218);
//---- значение 0 отображаться не будет
    SetIndexEmptyValue(0,0.0);
    SetIndexEmptyValue(1,0.0);
//---- отображение в DataWindow
    SetIndexLabel(0,"Fractal Up");
    SetIndexLabel(1,"Fractal Down");
//---- инициализация сделана
    return(0);
}
```

## 11.9 SetIndexLabel

`void SetIndexLabel(int index, string text)`

Установка имени линии индикатора для отображения информации в окне DataWindow и всплывающей подсказке.

### Параметры:

**index** - Порядковый номер линии индикатора. Должен быть от 0 до 8.

**text** - Текст описания линии индикатора. NULL означает, что значение этой линии не показывается в DataWindow.

### Пример:

```
//+-----+
//| Функция инициализации Ichimoku Kinko Hyo |
```

```

//+-----+
int init()
{
//---- линия Tenkan Sen
    SetIndexStyle(0,DRAW_LINE);
    SetIndexBuffer(0,Tenkan_Buffer);
    SetIndexDrawBegin(0,Tenkan-1);
    SetIndexLabel(0,"Tenkan Sen");
//---- линия Kijun Sen
    SetIndexStyle(1,DRAW_LINE);
    SetIndexBuffer(1,Kijun_Buffer);
    SetIndexDrawBegin(1,Kijun-1);
    SetIndexLabel(1,"Kijun Sen");
//---- гистограмма облака Kumo Up
    a_begin=Kijun; if(a_begin<Tenkan) a_begin=Tenkan;
    SetIndexStyle(2,DRAW_HISTOGRAM,STYLE_DOT);
    SetIndexBuffer(2,SpanA_Buffer);
    SetIndexDrawBegin(2,Kijun+a_begin-1);
    SetIndexShift(2,Kijun);
//---- в DataWindow не показываем, так как есть ограничивающая линия Senkou
Span A
    SetIndexLabel(2,NULL);
//---- линия Senkou Span A
    SetIndexStyle(5,DRAW_LINE,STYLE_DOT);
    SetIndexBuffer(5,SpanA2_Buffer);
    SetIndexDrawBegin(5,Kijun+a_begin-1);
    SetIndexShift(5,Kijun);
    SetIndexLabel(5,"Senkou Span A");
//---- гистограмма облака Kumo Down
    SetIndexStyle(3,DRAW_HISTOGRAM,STYLE_DOT);
    SetIndexBuffer(3,SpanB_Buffer);
    SetIndexDrawBegin(3,Kijun+Senkou-1);
    SetIndexShift(3,Kijun);
//---- в DataWindow не показываем, так как есть ограничивающая линия Senkou
Span B
    SetIndexLabel(3,NULL);
//---- линия Senkou Span B
    SetIndexStyle(6,DRAW_LINE,STYLE_DOT);
    SetIndexBuffer(6,SpanB2_Buffer);
    SetIndexDrawBegin(6,Kijun+Senkou-1);
    SetIndexShift(6,Kijun);
    SetIndexLabel(6,"Senkou Span B");
//---- линия Chinkou Span
    SetIndexStyle(4,DRAW_LINE);
    SetIndexBuffer(4,Chinkou_Buffer);
    SetIndexShift(4,-Kijun);
    SetIndexLabel(4,"Chinkou Span");
//----
    return(0);
}

```

## 11.10 SetIndexShift

```
void SetIndexLabel(int index, string text)
```

Установка имени линии индикатора для отображения информации в окне DataWindow и всплывающей подсказке.

### Параметры:

- index** - Порядковый номер линии индикатора. Должен быть от 0 до 7.
- text** - Текст описания линии индикатора. NULL означает, что значение этой линии не показывается в DataWindow.

### Пример:

```
//+-----+
//|  Функция инициализации Ichimoku Kinko Hyo      |
//+-----+
int init()
{
//---- линия Tenkan Sen
    SetIndexStyle(0, DRAW_LINE);
    SetIndexBuffer(0, Tenkan_Buffer);
    SetIndexDrawBegin(0, Tenkan-1);
    SetIndexLabel(0, "Tenkan Sen");
//---- линия Kijun Sen
    SetIndexStyle(1, DRAW_LINE);
    SetIndexBuffer(1, Kijun_Buffer);
    SetIndexDrawBegin(1, Kijun-1);
    SetIndexLabel(1, "Kijun Sen");
//---- гистограмма облака Kumo Up
    a_begin=Kijun; if(a_begin<Tenkan) a_begin=Tenkan;
    SetIndexStyle(2, DRAW_HISTOGRAM, STYLE_DOT);
    SetIndexBuffer(2, SpanA_Buffer);
    SetIndexDrawBegin(2, Kijun+a_begin-1);
    SetIndexShift(2, Kijun);
//---- в DataWindow не показываем, так как есть ограничивающая линия Senkou
Span A
    SetIndexLabel(2, NULL);
//---- линия Senkou Span A
    SetIndexStyle(5, DRAW_LINE, STYLE_DOT);
    SetIndexBuffer(5, SpanA2_Buffer);
    SetIndexDrawBegin(5, Kijun+a_begin-1);
    SetIndexShift(5, Kijun);
    SetIndexLabel(5, "Senkou Span A");
//---- гистограмма облака Kumo Down
    SetIndexStyle(3, DRAW_HISTOGRAM, STYLE_DOT);
    SetIndexBuffer(3, SpanB_Buffer);
    SetIndexDrawBegin(3, Kijun+Senkou-1);
    SetIndexShift(3, Kijun);
//---- в DataWindow не показываем, так как есть ограничивающая линия Senkou
Span B
    SetIndexLabel(3, NULL);
//---- линия Senkou Span B
```

```

SetIndexStyle(6, DRAW_LINE, STYLE_DOT);
SetIndexBuffer(6, SpanB2_Buffer);
SetIndexDrawBegin(6, Kijun+Senkou-1);
SetIndexShift(6, Kijun);
SetIndexLabel(6, "Senkou Span B");
//---- линия Chinkou Span
SetIndexStyle(4, DRAW_LINE);
SetIndexBuffer(4, Chinkou_Buffer);
SetIndexShift(4, -Kijun);
SetIndexLabel(4, "Chinkou Span");
//----
return(0);
}

```

### 11.11 SetIndexStyle

```

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

```

Устанавливает новый тип, стиль, ширину и цвет для указанной линии индикатора.

**Параметры:**

- index** - Порядковый номер линии. Должен быть от 0 до 7.
- type** - Стиль отрисовки линии индикатора. Может быть одним из перечисленных стилей отрисовки линии.
- style** - Стиль линии. Используется для линий толщиной в 1 пиксель. Может быть одним из перечисленных стилей линии. Пустое значение (EMPTY) указывает, что стиль не будет изменен.
- width** - Ширина линии. Допустимые значения - 1,2,3,4,5. Пустое значение (EMPTY) указывает, что ширина не будет изменена.
- clr** - Цвет линии. Отсутствие параметра означает, что цвет не будет изменен.

**Пример:**

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

### 11.12 SetLevelStyle

```

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

```

Устанавливает новый стиль, ширину и цвет для горизонтальных уровней индикатора, выводимого в отдельное окно.

**Параметры:**

- draw\_style** - Стиль линии. Может быть одним из перечисленных стилей линии. Пустое значение (EMPTY) указывает, что стиль не будет изменен.
- line\_width** - Ширина линии. Допустимые значения - 1,2,3,4,5. Пустое значение (EMPTY) указывает, что ширина не будет изменена.
- clr** - Цвет линии. Пустое значение CLR\_NONE указывает, что цвет не будет изменен.

**Пример:**

```
//---- уровни в виде толстой красной линии  
SetLevelStyle(STYLE_SOLID,2,Red)
```

### **11.13 SetLevelValue**

**void SetLevelValue(int level, double value)**

Устанавливает значение для указанного горизонтального уровня индикатора, выводимого в отдельное окно.

**Параметры:**

- level** - Номер уровня (0-31).
- value** - Значение для указанного уровня.

**Пример:**

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

## **12 Дата и время**

Группа функций, обеспечивающих работу с данными типа `datetime` (целое число, представляющее собой количество секунд, прошедших с 0 часов 1 января 1970 года).

### **12.1 Day**

**`int Day()`**

Возвращает текущий день месяца, т.е день месяца последнего известного времени сервера.

Замечание: при тестировании последнее известное время сервера моделируется.

**Пример:**

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

### **12.2 DayOfWeek**

**`int DayOfWeek()`**

Возвращает порядковый номер дня недели (воскресенье-0,1,2,3,4,5,6) последнего известного времени сервера.

Замечание: при тестировании последнее известное время сервера моделируется.

**Пример:**

```
// не работает в выходные дни.  
if(DayOfWeek() $==$ 0 || DayOfWeek() $==$ 6) return(0);
```

### **12.3 DayOfYear**

**`int DayOfYear()`**

Возвращает текущий день года (1-1 января,...,365(6) - 31 декабря), т.е день года последнего известного времени сервера.

Замечание: при тестировании последнее известное время сервера моделируется.

**Пример:**

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

### **12.4 Hour**

**`int Hour()`**

Возвращает текущий час (0,1,2,..23) последнего известного серверного времени на момент старта программы (в процессе выполнения программы это значение не меняется).

Замечание: при тестировании последнее известное время сервера моделируется.

**Пример:**

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

## **12.5 Minute**

**int Minute()**

Возвращает текущую минуту (0,1,2,..59) последнего известного серверного времени на момент старта программы (в процессе выполнения программы это значение не меняется).

Замечание: при тестировании последнее известное время сервера моделируется.

**Пример:**

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

## **12.6 Month**

**int Month()**

Возвращает номер текущего месяца (1-Январь,2,3,4,5,6,7,8,9,10,11,12), т.е. номер месяца последнего известного времени сервера.

Замечание: при тестировании последнее известное время сервера моделируется.

**Пример:**

```
if(Month()<=5)
    return("первое полугодие");
```

## **12.7 Seconds**

**int Seconds()**

Возвращает количество секунд, прошедших с начала текущей минуты последнего известного серверного времени на момент старта программы (в процессе выполнения программы это значение не меняется).

Замечание: при тестировании последнее известное время сервера моделируется.

**Пример:**

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

## **12.8 TimeCurrent**

**datetime TimeCurrent()**

Возвращает последнее известное время сервера (время прихода последней котировки) в виде количества секунд, прошедших после 00:00 1 января 1970 года.

Замечание: при тестировании последнее известное время сервера моделируется.

**Пример:**

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

## **12.9 TimeDay**

**int TimeDay(datetime date)**

Возвращает день месяца (1 - 31) для указанной даты.

**Параметры:**

**date** - Дата, представленная в виде количества секунд, прошедших после 00:00 1 января 1970 года.

**Пример:**

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

## **12.10 TimeDayOfWeek**

**int TimeDayOfWeek(datetime date)**

Возвращает день недели (0-Воскресенье,1,2,3,4,5,6) для указанной даты.

**Параметры:**

**date** - Дата, представленная в виде количества секунд, прошедших после 00:00 1 января 1970 года.

**Пример:**

```
int weekday=TimeDayOfWeek(D'2004.11.2');  
// день 2 - вторник
```

## **12.11 TimeDayOfYear**

**int TimeDayOfYear(datetime date)**

Возвращает день (1 - 1 января,...,365(6) - 31 декабря) года для указанной даты.

**Параметры:**

**date** - Дата, представленная в виде количества секунд, прошедших после 00:00 1 января 1970 года.

**Пример:**

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

### **12.12 TimeHour**

**int TimeHour(datetime time)**

Возвращает час для указанного времени.

**Параметры:**

**time** - Дата, представленная в виде количества секунд, прошедших после 00:00 1 января 1970 года.

**Пример:**

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

### **12.13 TimeLocal**

**datetime TimeLocal()**

Возвращает локальное компьютерное время в виде количества секунд, прошедших после 00:00 1 января 1970 года.

Замечание: при тестировании локальное время моделируется и совпадает с моделированным последним известным временем сервера.

**Пример:**

```
if(TimeLocal()-OrderOpenTime() $<$ 360) return(0);
```

### **12.14 TimeMinute**

**int TimeMinute(datetime time)**

Возвращает минуты для указанного времени.

**Параметры:**

**time** - Дата, представленная в виде количества секунд, прошедших после 00:00 1 января 1970 года.

**Пример:**

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

### **12.15 TimeMonth**

```
int TimeMonth(datetime time)
```

Возвращает номер месяца для указанного времени (1-Январь,2,3,4,5,6,7,8,9,10,11,12).

#### **Параметры:**

**time** - Дата, представленная в виде количества секунд, прошедших после 00:00 1 января 1970 года.

#### **Пример:**

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

### **12.16 TimeSeconds**

```
int TimeSeconds(datetime time)
```

Возвращает количество секунд, прошедших с начала минуты для указанного времени.

#### **Параметры:**

**time** - Дата, представленная в виде количества секунд, прошедших после 00:00 1 января 1970 года.

#### **Пример:**

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

### **12.17 TimeYear**

```
int TimeYear(datetime time)
```

Возвращает год для указанной даты. Возвращаемая величина может быть в диапазоне 1970-2037.

#### **Параметры:**

**time** - Дата, представленная в виде количества секунд, прошедших после 00:00 1 января 1970 года.

#### **Пример:**

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

## **12.18 Year**

**int Year()**

Возвращает текущий год, т.е. год последнего известного времени сервера.

Замечание: при тестировании последнее известное время сервера моделируется.

**Пример:**

```
// возврат, если дата находится в диапазоне от 1 января до 30 апреля 2006
года.
if(Year()==2006 && Month() $<$ 5)
    return(0);
```



## 13 Файловые операции

Группа функций для работы с файлами.

Существует три каталога (с подкаталогами), в которых могут располагаться рабочие файлы:

- /HISTORY/<текущий брокер> - специально для функции FileOpenHistory;
- /EXPERTS/FILES - общий случай;
- /TESTER/FILES - специально для тестирования.

Работа с файлами из других каталогов пресекается.

### 13.1 FileClose

**void FileClose(int handle)**

Закрытие файла, ранее открытого функцией FileOpen().

**Параметры:**

**handle** - Файловый дескриптор, возвращаемый функцией FileOpen().

**Пример:**

```
int handle=FileOpen("имя файла", FILE_CSV|FILE_READ);
if(handle>0)
{
    // работает с файлом ...
    FileClose(handle);
}
```

### 13.2 FileDelete

**void FileDelete(string filename)**

Удаление указанного файла. Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

Файлы могут быть удалены только в том случае, если они расположены в папке *каталог\_терминала\experts\files* (*каталог\_терминала\tester\files* в случае тестирования эксперта) или ее подпапках.

**Параметры:**

**filename** - Имя файла.

**Пример:**

```
// файл my_table.csv будет удален из папки terminal_dir\experts\files
```

```

int lastError;
FileDelete("my_table.csv");
lastError=GetLastError();
if(lastError!=ERR_NOERROR)
{
    Print("ошибка (" ,lastError,") при удалении файла my_table.csv");
    return(0);
}

```

### **13.3 FileFlush**

**void FileFlush(int handle)**

Сброс на диск всех данных, оставшихся в файловом буфере ввода-вывода.

Замечания: функцию FileFlush() необходимо вызывать между операциями чтения из файла и записи в файл.

При закрытии файла данные сбрасываются на диск автоматически, поэтому нет необходимости вызывать функцию FileFlush() перед вызовом функции FileClose().

**Параметры:**

**handle** - Файловый дескриптор, возвращаемый функцией FileOpen().

**Пример:**

```

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

```

### **13.4 FileIsEnding**

**bool FileIsEnding(int handle)**

Возвращает TRUE, если файловый указатель находится в конце файла, иначе возвращает FALSE.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError(). В случае достижения конца файла в процессе чтения функция GetLastError() вернет ошибку ERR\_END\_OF\_FILE (4099).

### Параметры:

**handle** - Файловый дескриптор, возвращаемый функцией `FileOpen()`.

### Пример:

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

## 13.5 FileIsLineEnding

**bool FileIsLineEnding(int handle)**

Возвращает TRUE, если файловый указатель находится в конце строки файла формата CSV, иначе возвращает FALSE. Чтобы получить информацию об ошибке, необходимо вызвать функцию `GetLastError()`.

### Параметры:

**handle** - Файловый дескриптор, возвращаемый функцией `FileOpen()`.

### Пример:

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

## 13.6 FileOpen

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

Открывает Файл для ввода и/или вывода. Возвращает файловый дескриптор открытого файла или -1 в случае неудачи. Чтобы получить дополнительную информацию об ошибке, необходимо вызвать функцию `GetLastError()`.

Замечания: файлы могут открываться только в папке *каталог\_терминала\experts\files* (*каталог\_терминала\tester\files* в случае тестирования эксперта) или ее подпапках.

Нельзя одновременно использовать режимы FILE\_BIN и FILE\_CSV.

Если FILE\_WRITE не комбинируется с FILE\_READ, то будет открыт файл нулевой длины. Даже если до открытия в файле были данные, то они будут уничтожены. Если необходимо дописывать данные в существующий файл, то его нужно открывать, используя комбинацию FILE\_READ | FILE\_WRITE.

Если FILE\_READ не комбинируется с FILE\_WRITE, то файл будет открыт только в том

случае, если он уже существует. Если файл не существует, то создать его можно, используя режим FILE\_WRITE.

Одновременно может быть открыто не более 32 файлов в пределах одного исполняемого модуля. Описатели файлов, открытых в одном модуле, нельзя передавать в другие модули (библиотеки).

#### Параметры:

**filename** - Имя файла.

**mode** - Способ открытия. Это может быть одна величина или их комбинация: FILE\_BIN, FILE\_CSV, FILE\_READ, FILE\_WRITE.

**delimiter** - Знак разделителя для csv-файлов. По умолчанию применяется символ ';'.

#### Пример:

```
int handle;
handle=FileOpen("my_data.csv",FILE_CSV|FILE_READ, ';');
if(handle<1)
{
    Print("Файл my_data.dat не обнаружен, последняя ошибка ",
GetLastError());
    return(false);
}
```

### **13.7 FileOpenHistory**

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

Открывает файл в текущей папке истории (*каталог\_терминала\history\server\_name*) или ее подпапках. Возвращает описатель файла или -1 в случае неудачи. Для получения дополнительной информации об ошибке необходимо вызвать функцию GetLastError().

Замечания: клиентский терминал может подключаться к серверам разных брокерских компаний. Исторические данные (файлы HST) каждой брокерской компании хранятся в соответствующей подпапке папки истории *каталог\_терминала\history*.

Функция может быть полезна для формирования собственных исторических данных нестандартного символа и/или периода. Файл, сформированный в папке истории может быть открыт автономно, для его графика не требуется подкачка данных.

#### Параметры:

**filename** - Имя файла.



### **13.9 FileReadDouble**

**double FileReadDouble(int handle, int size=DOUBLE\_VALUE)**

Функция читает число двойной точности с плавающей точкой (double) из текущей позиции бинарного файла. Размер числа может быть 8 байтов (double) или 4 байта (float). Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

#### **Параметры:**

- handle** - Файловый дескриптор, возвращаемый функцией FileOpen().
- size** - Формат числа. Может быть DOUBLE\_VALUE (8 байтов) или FLOAT\_VALUE (4 байта).

#### **Пример:**

```
int handle;
double value;
handle=FileOpen("mydata.dat", FILE_BIN);
if(handle>0)
{
    value=FileReadDouble(handle, DOUBLE_VALUE);
    FileClose(handle);
}
```

### **13.10 FileReadInteger**

**int FileReadInteger(int handle, int size=LONG\_VALUE)**

Функция читает целое число из текущей позиции бинарного файла. Размер целого числа может быть 1, 2 или 4 байта. Если размер числа не указан, система пытается прочитать как 4-байтовое целое число.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

#### **Параметры:**

- handle** - Файловый дескриптор, возвращаемый функцией FileOpen().
- size** - Формата числа. Может быть CHAR\_VALUE(1 байт), SHORT\_VALUE(2 байта) или LONG\_VALUE(4 байта).

#### **Пример:**

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

### 13.11 FileReadNumber

**double FileReadNumber(int handle)**

Чтение числа с текущей позиции файла CSV до разделителя. Применяется только для файлов CSV.

Для того чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

#### Параметры:

**handle** - Файловый дескриптор, возвращаемый функцией FileOpen().

#### Пример:

```
int handle;
int value;
handle=FileOpen("filename.csv", FILE_CSV, '\t');
if(handle>0)
{
    value=FileReadNumber(handle);
    FileClose(handle);
}
```

### 13.12 FileReadString

**string FileReadString(int handle, int length=0)**

Функция читает строку с текущей позиции файла. Применяется как к CSV, так и к двоичным файлам. Для текстовых файлов строка будет прочитана до разделителя. Для бинарных файлов в строку будет прочитано указанное количество символов.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

#### Параметры:

**handle** - Файловый дескриптор, возвращаемый функцией FileOpen().

**length** - Количество символов для чтения.

#### Пример:

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

### 13.13 FileSeek

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

Функция перемещает файловый указатель на новую позицию, которая является смещением в байтах от начала, конца или текущей позиции файла. Следующее чтение или запись происходят с новой позиции.

Если перемещение файлового указателя прошло успешно, функция возвращает TRUE, иначе возвращает FALSE. Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

#### Параметры:

**handle** - Файловый дескриптор, возвращаемый функцией FileOpen().

**offset** - Смещение в байтах.

**origin** - Начальное положение. Величина может быть одной из этих констант:  
SEEK\_CUR - от текущего положения,  
SEEK\_SET - от начала,  
SEEK\_END - от конца файла.

#### Пример:

```
int handle=FileOpen("filename.csv", FILE_CSV|FILE_READ|FILE_WRITE, ';');
if(handle>0)
{
    FileSeek(handle, 0, SEEK_END);
    //---- add data to the end of file
    FileWrite(handle, data1, data2);
    FileClose(handle);
    handle=0;
}
```

### 13.14 FileSize

**int FileSize(int handle)**

Функция возвращает размер файла в байтах.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

#### Параметры:

**handle** - Файловый дескриптор, возвращаемый функцией FileOpen().

#### Пример:

```
int handle;
int size;
```

```

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

```

### 13.15 FileTell

**int FileTell(int handle)**

Функция возвращает смещение текущей позиции файлового указателя от начала файла.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

**Параметры:**

**handle** - Файловый описатель, возвращаемый функцией FileOpen().

**Пример:**

```

int handle;
int pos;
handle=FileOpen("my_table.dat", FILE_BIN|FILE_READ);
// чтение некоторых данных
pos=FileTell(handle);
Print("current position is ", pos);

```

### 13.16 FileWrite

**int FileWrite(int handle, ...)**

Функция предназначена для записи данных в файл CSV, разделитель между данными включается автоматически. После записи в файл добавляется признак конца строки "\r\n".

При выводе числовые данные преобразуются в текстовый формат (см. функцию Print()).

Возвращает количество записанных символов или отрицательное значение, если происходит ошибка .

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

**Параметры:**

**handle** - Файловый описатель, возвращаемый функцией FileOpen().

**...** - Данные, разделенные запятыми. Может быть не больше 63 параметров.

Данные типов double, int автоматически преобразовываются в строку, данные типов color, datetime и bool воспринимаются как

целые числа (тип `int`), данные типа `string` выводятся как есть, без преобразования.

В качестве параметра нельзя передать массивы, массивы могут быть выведены поэлементно.

**Пример:**

```
int handle;
datetime orderOpen=OrderOpenTime();
handle=FileOpen("filename", FILE_CSV|FILE_WRITE, '\t');
if(handle>0)
{
    FileWrite(handle, Close[0], Open[0], High[0], Low[0],
TimeToStr(orderOpen));
    FileClose(handle);
}
```

### **13.17 FileWriteArray**

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

Функция записывает массив в бинарный файл. Массивы типа `int`, `bool`, `datetime` и `color` записываются поэлементно как 4-байтовые целые числа. Массивы типа `double` записываются поэлементно как 8-байтовые числа с плавающей запятой. Массивы типа `string` записываются построчно, после каждой строки автоматически добавляется признак конца строки `"\r\n"`.

Возвращает число записанных элементов или отрицательное значение в случае ошибки.

Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

**Параметры:**

- handle** - Файловый дескриптор, возвращаемый функцией `FileOpen()`.
- array[]** - Массив для записи.
- start** - Начальный индекс в массиве (номер первого записываемого элемента).
- count** - Количество записываемых элементов.

**Пример:**

```
int handle;
double BarOpenValues[10];
// скопирует первые десять баров в массив
for(int i=0;i<10; i++)
    BarOpenValues[i]=Open[i];
```

```

// запись массива в файл
handle=FileOpen("mydata.dat", FILE_BIN|FILE_WRITE);
if(handle>0)
{
    FileWriteArray(handle, VarOpenValues, 3, 7); // запись последних 7
элементов
    FileClose(handle);
}

```

### 13.18 FileWriteDouble

```

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

```

Функция записывает число с плавающей запятой в двоичный файл. Если формат задан `FLOAT_VALUE`, то значение будет записано как 4-байтовое число с плавающей запятой (тип `float`), иначе будет записан в 8-байтовом формате с плавающей запятой (тип `double`). Возвращает фактически записанное число байт или отрицательное значение в случае ошибки.

Для того чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

#### Параметры:

**handle** - Файловый дескриптор, возвращаемый функцией `FileOpen()`.

**value** - Значение двойной точности.

**size** - Дополнительный флаг формата. Может быть любой из следующих величин:

- `DOUBLE_VALUE` (8 байтов, по умолчанию)
- `FLOAT_VALUE` (4 байта).

#### Пример:

```

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

```

### **13.19 FileWriteInteger**

**int FileWriteInteger(int handle, int value, int size=LONG\_VALUE)**

Функция записывает значение целого числа в двоичный файл. Если размер - SHORT\_VALUE, значение будет записано как 2-байтовое целое число (тип short), если размер - CHAR\_VALUE, значение будет записано как 1-байтовое целое число (тип char), если размер - LONG\_VALUE, значение будет записано как 4-байтовое целое число (тип long int).

Возвращает число фактически записанных байтов или отрицательное значение в случае ошибки.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

#### **Параметры:**

- handle** - Файловый дескриптор, возвращаемый функцией FileOpen().
- value** - Значение для записи.
- size** - Дополнительный флаг формата. Может быть любой из следующих величин:
  - CHAR\_VALUE (1 байт),
  - SHORT\_VALUE (2 байта),
  - LONG\_VALUE (4 байта, по умолчанию).

#### **Пример:**

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

### **13.20 FileWriteString**

**int FileWriteString(int handle, string value, int size)**

Функция записывает строку в двоичный файл с текущей позиции.

Возвращает число фактически записанных байтов или отрицательное значение в случае ошибки.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

### Параметры:

- handle** - Файловый дескриптор, возвращаемый функцией FileOpen().
- value** - Записываемая строка.
- size** - Длина записываемой строки. Если строка длиннее, чем указанное значение, то она будет усечена. Если строка короче, то она будет дополнена двоичными 0 до указанной длины.

### Пример:

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

## 14 Глобальные переменные

Группа функций, предназначенных для работы с глобальными переменными.

Не следует путать глобальные переменные клиентского терминала с переменными, объявленными на глобальном уровне MQL4-программы.

Глобальные переменные существуют в клиентском терминале 4 недели с момента последнего обращения, после этого автоматически уничтожаются. Обращением к глобальной переменной считается не только установка нового значения, но и чтение значения глобальной переменной.

Глобальные переменные клиентского терминала доступны одновременно из всех MQL4-программ, запущенных на клиентском терминале

### 14.1 GlobalVariableCheck

**bool GlobalVariableCheck(string name)**

Возвращает значение TRUE, если глобальная переменная существует, иначе возвращает FALSE.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

**Параметры:**

**name** - Имя глобальной переменной.

**Пример:**

```
// контроль переменной перед использованием
if(!GlobalVariableCheck("g1"))
    GlobalVariableSet("g1",1);
```

### 14.2 GlobalVariableDel

**bool GlobalVariableDel(string name)**

Удаляет глобальную переменную. При успешном удалении функция возвращает TRUE, иначе FALSE. Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

**Параметры:**

**name** - Имя глобальной переменной.

**Пример:**

```
// удаление глобальной переменной с именем "gvar_1"
GlobalVariableDel("gvar_1");
```

### **14.3 GlobalVariableGet**

**double GlobalVariableGet(string name)**

Возвращает значение существующей глобальной переменной или 0 в случае ошибки.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

**Параметры:**

**name** - Имя глобальной переменной.

**Пример:**

```
double v1=GlobalVariableGet("g1");
//---- проверьте результат запроса функции
if(GetLastError()!=0) return(false);
//---- продолжение обработки
```

### **14.4 GlobalVariableName**

**string GlobalVariableName(int index)**

Функция возвращает имя глобальной переменной по порядковому номеру в списке глобальных переменных. Чтобы получить информацию об ошибке необходимо вызвать функцию функцию GetLastError().

**Параметры:**

**index** - Порядковый номер в списке глобальных переменных. Должен быть большим или равным 0 и меньшим, чем GlobalVariablesTotal().

**Пример:**

```
int    var_total=GlobalVariablesTotal();
string name;
for(int i=0;i<var_total;i++)
{
    name=GlobalVariableName(i);
    Print(i,": Имя глобальной переменной - ",name);
}
```

### **14.5 GlobalVariableSet**

**datetime GlobalVariableSet(string name, double value)**

Устанавливает новое значение глобальной переменной. Если переменная не существует, то система создает новую глобальную переменную. При успешном выполнении функция

возвращает время последнего доступа, иначе 0. Чтобы получить информацию об ошибке, необходимо вызвать функцию `GetLastError()`.

#### Параметры:

**name** - Имя глобальной переменной.

**value** - Новое числовое значение.

#### Пример:

```
//---- попытка устанавливать новое значение
if(GlobalVariableSet("BarsTotal",Bars)==0)
    return(false);
//---- продолжение обработки
```

### **14.6 GlobalVariableSetOnCondition**

**bool** `GlobalVariableSetOnCondition`( **string name**, **double value**,  
**double check\_value**)

Устанавливает новое значение существующей глобальной переменной, если текущее значение переменной равно значению третьего параметра *check\_value*. Если переменной не существует, функция сгенерирует ошибку `ERR_GLOBAL_VARIABLE_NOT_FOUND` (4058) и вернет `FALSE`. При успешном выполнении функция возвращает `TRUE`, иначе `FALSE`. Для того, чтобы получить информацию об ошибке, необходимо вызвать функцию `GetLastError()`. Если текущее значение глобальной переменной отличается от *check\_value*, функция вернет `FALSE`.

Функция обеспечивает атомарный доступ к глобальной переменной, поэтому она может быть использована для организации семафора при взаимодействии нескольких одновременно работающих экспертов в пределах одного клиентского терминала.

#### Параметры:

**name** - Имя глобальной переменной.

**value** - Новое значение.

**check\_value** - Значение для проверки текущего значения глобальной переменной.

#### Пример:

```
int init()
{
    //---- создание глобальной переменной
    GlobalVariableSet("DATAFILE_SEM",0);
    //...
}
```

```

int start()
{
    //---- перед использованием ресурса пытаемся его заблокировать
    while(!IsStopped())
    {
        //---- блокируем
        if(GlobalVariableSetOnCondition("DATAFILE_SEM",1,0)==true) break;
        //---- переменная удалена?
        if(GetLastError()==ERR_GLOBAL_VARIABLE_NOT_FOUND) return(0);
        //---- задержка исполнения в полсекунды
        Sleep(500);
    }
    //---- ресурс заблокирован
    // ... работа с файлом
    //---- разблокируем ресурс
    GlobalVariableSet("file_semaphore",0);
}

```

### **14.7 GlobalVariablesDeleteAll**

**int GlobalVariablesDeleteAll(string prefix\_name=NULL)**

Удаляет глобальные переменные. Если префикс для имени не задан, то удаляются все глобальные переменные. В противном случае удаляются только те переменные, имена которых начинаются на указанный префикс. Функция возвращает количество удаленных переменных.

#### **Параметры:**

**prefix\_name** - Префикс имени удаляемых глобальных переменных.

#### **Пример:**

```
Print("Удалено ",GlobalVariablesDeleteAll("test_")," глобальных переменных
после тестирования");
```

### **14.8 GlobalVariablesTotal**

**int GlobalVariablesTotal()**

Функция возвращает общее количество глобальных переменных.

#### **Пример:**

```
Print("В клиентском терминале ",GlobalVariablesTotal()," глобальных
переменных");
```

## 15 Математические функции

Набор математических и тригонометрических функций.

### 15.1 *MathAbs*

**double MathAbs(double value)**

Функция возвращает абсолютное значение (значение по модулю) переданного ей числа

**Параметры:**

**value** - Числовая величина.

**Пример:**

```
double dx=-3.141593, dy;
// вычисляет MathAbs
dy=MathAbs(dx);
Print("Абсолютная величина ",dx," есть ",dy);
// Вывод: абсолютная величина -3.141593 есть 3.141593
```

### 15.2 *MathArccos*

**double MathArccos(double x)**

Функция возвращает значение арккосинуса  $x$  в диапазоне 0 к  $\pi$  в радианах. Если  $x$  меньше -1 или больше 1, функция возвращает NaN (неопределенное значение).

**Параметры:**

**x** - Значение между -1 и 1, арккосинус которого должен быть вычислен.

**Пример:**

```
double x=0.32696, y;
y=asin(x);
Print("арксинус от ",x," = ",y);
y=acos(x);
Print("арккосинус от ",x," = ",y);
// Вывод: арксинус от 0.326960=0.333085
// Вывод: арккосинус от 0.326960=1.237711
```

### 15.3 *MathArcsin*

**double MathArcsin(double x)**

Функция возвращает арксинус  $x$  в диапазоне от  $-\pi/2$  до  $\pi/2$  радианов. Если  $x$ -, меньше -1 или больше 1, функция возвращает NaN (неопределенное значение).

**Параметры:**

**x** - Значение, для которого должен быть вычислен арксинус.

### Пример:

```
double x=0.32696, y;  
y=asin(x);  
Print("арксинус от ",x," = ",y);  
y=acos(x);  
Print("арккосинус от ",x," = ",y);  
// Вывод: арксинус от 0.326960=0.333085  
// Вывод: арккосинус от 0.326960=1.237711
```

## 15.4 MathArctan

**double MathArctan(double x)**

Функция возвращает арктангенс  $x$ . Если  $x$  равен 0, функция возвращает 0. MathArctan возвращает значение в диапазоне от  $-\pi/2$  до  $\pi/2$  радианов.

### Параметры:

**x** - Число, представляющее тангенс.

### Пример:

```
double x=-862.42, y;  
y=MathArctan(x);  
Print("Арктангенс от ",x," есть ",y);  
// Вывод: Арктангенс от -862.42 есть -1.5696
```

## 15.5 MathCeil

**double MathArctan(double x)**

Функция возвращает арктангенс  $x$ . Если  $x$  равен 0, функция возвращает 0. MathArctan возвращает значение в диапазоне от  $-\pi/2$  до  $\pi/2$  радианов.

### Параметры:

**x** - Число, представляющее тангенс.

### Пример:

```
double x=-862.42, y;  
y=MathArctan(x);  
Print("Арктангенс от ",x," есть ",y);  
// Вывод: Арктангенс от -862.42 есть -1.5696
```

## 15.6 MathCos

**double MathCos(double value)**

Функция возвращает косинус угла.

### Параметры:

**value** - Угол в радианах.

**Пример:**

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

## **15.7 MathExp**

**double MathExp(double d)**

Функция возвращает значение числа  $e$  в степени  $d$ . При переполнении функция возвращает INF (бесконечность), в случае потери порядка MathExp возвращает 0.

**Параметры:**

**d** - Число, определяющее степень.

**Пример:**

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

## **15.8 MathFloor**

**double MathFloor(double x)**

Функция возвращает числовое значение, представляющее наибольшее целое число, которое меньше или равно  $x$ .

**Параметры:**

**x** - Числовое значение.

**Пример:**

```
double y;
y=MathFloor(2.8);
Print("Наименьшее целое от 2.8 есть ",y);
y=MathFloor(-2.8);
Print("Наименьшее целое от -2.8 есть ",y);
/*Вывод:
   Наименьшее целое от 2.8 есть 2
   Наименьшее целое от -2.8 есть -3*/
```

## 15.8 MathLog

**double MathLog(double x)**

Функции возвращают натуральный логарифм  $x$  в случае успеха. Если  $x$  отрицателен, функция возвращает NaN (неопределенное значение). Если  $x$  равен 0, функция возвращает INF (бесконечность).

**Параметры:**

**x** - Значение, логарифм которого должен быть вычислен.

**Пример:**

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

## 15.9 MathMax

**double MathMax(double value1, double value2)**

Функция возвращает максимальное из двух числовых значений.

**Параметры:**

**value1** - Первое числовое значение.

**value2** - Второе числовое значение.

**Пример:**

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

## 15.10 MathMin

**double MathMin(double value1, double value2)**

Функция возвращает минимальное из двух числовых значений.

**Параметры:**

**value1** - Первое числовое значение.

**value2** - Второе числовое значение.

**Пример:**

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

### 15.11 MathMod

**double MathMod(double value, double value2)**

Функция возвращает вещественный остаток от деления двух чисел.

Функция MathMod рассчитывает вещественный остаток  $f$  от  $x/y$  таким образом, что  $x = i * y + f$ , где  $i$  является целым числом,  $f$  имеет тот же знак, что и  $x$ , и абсолютное значение  $f$  меньше, чем абсолютное значение  $y$ .

**Параметры:**

**value** - Значение делимого.

**value2** - Значение делителя.

**Пример:**

```
double x=-10.0, y=3.0, z;  
z=MathMod(x, y);  
Print("Остаток от ", x, " / ", y, " есть ", z);  
// Вывод: Остаток от -10 / 3 есть -1
```

### 15.12 MathPow

**double MathPow(double base, double exponent)**

Функция возвращает значение основания, возведенного в указанную степень.

**Параметры:**

**base** - Основание.

**exponent** - Значение степени.

**Пример:**

```
double x=2.0, y=3.0, z;  
z=MathPow(x, y);  
Printf(x, " в степени ", y, " есть ", z);  
// Вывод: 2 в степени 3 есть 8
```

### 15.13 MathRand

**int MathRand()**

Функция возвращает псевдослучайное целое число в диапазоне от 0 до 32767. Перед первым вызовом функции необходимо использовать функцию MathSrand, чтобы перевести генератор псевдослучайных чисел в начальное состояние.

**Пример:**

```
MathSrand(TimeLocal());
// Отображает 10 чисел.
for(int i=0;i<10;i++ )
    Print("произвольная величина ", MathRand());
```

### **15.14 MathRound**

**double MathRound(double value)**

Функция возвращает значение, округленное до ближайшего целого числа указанного числового значения.

**Параметры:**

**value** - Числовая величина для округления.

**Пример:**

```
double y=MathRound(2.8);
Print("Округление 2.8 до ",y);
y=MathRound(2.4);
Print("Округление -2.4 до ",y);
// Вывод: Округление 2.8 до 3
//         Округление -2.4 до -2
```

### **15.15 MathSin**

**double MathSin(double value)**

Функция возвращает синус указанного угла.

**Параметры:**

**value** - Угол в радианах.

**Пример:**

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

### **15.16 MathSqrt**

**double MathSqrt(double x)**

Функция возвращает квадратный корень  $x$ . Если  $x$  отрицателен, MathSqrt возвращает NaN (неопределенное значение).

**Параметры:**

**x** - Положительная числовая величина.

**Пример:**

```
double question=45.35, answer;
answer=MathSqrt(question);
if(question<0)
    Print("Ошибка: MathSqrt возвратил ",answer," ответ");
else
    Print("Квадратный корень из ",question," есть ", answer);
//Ответ: Квадратный корень из 45.35 есть 6.73
```

### **15.17 MathSrand**

**void MathSrand(int seed)**

Функция устанавливает начальное состояние для генерации ряда псевдослучайных целых чисел. Чтобы переинициализировать генератор (т.е. установить генератор в предыдущее начальное состояние), необходимо использовать значение 1 в качестве инициализирующего параметра. Любое другое значение для начального числа устанавливает генератор в случайную отправную точку. MathRand возвращает подряд сгенерированные псевдослучайные числа. Вызов MathRand перед любым вызовом MathSrand генерирует ту же самую последовательность, что и запрос MathSrand с параметром 1.

**Параметры:**

**seed** - Начальное число для ряда случайных чисел.

**Пример:**

```
MathSrand(TimeLocal());
// Отображает 10 чисел.
for(int i=0;i<10;i++ )
    Print("произвольная величина ", MathRand());
```

### **15.18 MathTan**

**double MathTan(double x)**

Функция возвращает тангенс  $x$ . Если  $x$  больше или равен 263 или меньше или равен -263, то происходит потеря значения и функция возвращает неопределенное число.

**Параметры:**

**x** - Угол в радианах.

**Пример:**

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

## 16 Графические объекты

Группа функций, предназначенных для работы с графическими объектами, относящимися к текущему графику.

### 16.1 ObjectCreate

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

Создание объекта с указанным именем, тип и начальные координаты в указанном подокне графика. Число координат, связываемых с объектом, может быть от 1 до 3 в зависимости от типа. При успешном создании объекта функция возвращает TRUE, иначе FALSE.

Чтобы получить дополнительную информацию об ошибке, необходимо вызвать функцию GetLastError().

Объекты с типом OBJ\_LABEL игнорируют координаты. Используйте функцию ObjectSet() для установки свойств OBJPROP\_XDISTANCE и OBJPROP\_YDISTANCE.

Замечания: нумерация подокон графика (если на графике есть подокна с индикаторами) начинается с 1. Главное окно графика есть всегда и имеет индекс 0.

Координаты должны передаваться парами - время и цена. Для примера, объекту OBJ\_VLINE требуется только время, но также нужно передать и цену (любое значение).

#### Параметры:

<b>name</b>	- Уникальное имя объекта.
<b>type</b>	- Тип объекта. Может быть любым из <u>типов объектов</u> .
<b>window</b>	- Индекс окна, в которое будет добавлен объект. Индекс окна должен быть большим или равным 0 и меньшим, чем <u>WindowsTotal()</u> .
<b>time1</b>	- Время первой координаты.
<b>price1</b>	- Цена первой координаты.
<b>time2</b>	- Время второй координаты.
<b>price2</b>	- Цена второй координаты.
<b>time3</b>	- Время третьей координаты.
<b>price3</b>	- Цена третьей координаты.

#### Пример:

```

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

```

## 16.2 ObjectDelete

**bool ObjectDelete(string name)**

Удаление объекта с указанным именем. При успешном удалении функция возвращает TRUE, иначе FALSE.

Чтобы получить дополнительную информацию об ошибке, необходимо вызвать функцию GetLastError().

**Параметры:**

**name** - Имя удаляемого объекта.

**Пример:**

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

## 16.3 ObjectDescription

**string ObjectDescription(string name)**

Функция возвращает описание объекта. Для объектов типа OBJ\_TEXT и OBJ\_LABEL возвращается текст, отображаемый этими объектами.

Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

См. также ObjectSetText().

**Параметры:**

**name** - Имя объекта.

**Пример:**

```

// сохранение списка объектов графика в файл
int handle, total;
string obj_name, fname;

```

```

// имя файла
fname="objlist_"+Symbol();
handle=FileOpen(fname,FILE_CSV|FILE_WRITE);
if(handle>0)
{
    total=ObjectsTotal();
    for(int i=-;i<total;i++)
    {
        obj_name=ObjectName(i);
        FileWrite(handle,"Object "+obj_name+" описание="
"+ObjectDescription(obj_name));
    }
    FileClose(handle);
}

```

## 16.4 ObjectFind

**int ObjectFind(string name)**

Поиск объекта с указанным именем. Функция возвращает индекс окна, которому принадлежит искомый объект. В случае неудачи функция вернет -1. Для получения дополнительной информации об ошибке необходимо вызвать функцию GetLastError().

Нумерация подокон графика (если на графике есть подокна с индикаторами) начинается с 1. Главное окно графика есть всегда и имеет индекс 0.

**Параметры:**

**name** - Имя искомого объекта.

**Пример:**

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

## 16.5 ObjectGet

**double ObjectGet(string name, int prop\_id)**

Функция возвращает значение указанного свойства объекта. Для получения информации об ошибке необходимо вызвать функцию GetLastError().

См. также ObjectSet().

**Параметры:**

**name** - Имя объекта.

**prop\_id** - Идентификатор свойства объекта. Может быть любым из значений списка свойств объекта.

**Пример:**

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

## 16.6 ObjectGetFiboDescription

**string ObjectGetFiboDescription(string name, int index)**

Функция возвращает описание уровня объекта Фибоначчи. Количество уровней зависит от типа объекта, принадлежащего к группе объектов Фибоначчи. Максимальное количество уровней - 32.

Для получения информации об ошибке необходимо вызвать функцию GetLastError().

См. также ObjectSetFiboDescription().

**Параметры:**

**name** - Имя объекта Фибоначчи.

**index** - Индекс уровня Фибоначчи (0-31).

**Пример:**

```
#include <stdlib.mqh>
...
string text;
for(int i=0;i<32;i++)
{
    text=ObjectGetFiboDescription(MyObjectName,i);
    //---- проверим, возможно уровней у объекта меньше, чем 32
    if(GetLastError()!=ERR_NO_ERROR) break;
    Print(MyObjectName,"номер уровня: ",i," описание: ",text);
}
```

## 16.7 ObjectGetShiftByValue

**int ObjectGetShiftByValue(string name, double value)**

Функция вычисляет и возвращает номер бара (смещение относительно текущего бара) для указанной цены. Номер бара вычисляется при помощи линейного уравнения по первой и второй координатам. Применяется для трендовых линий и аналогичных объектов. Для получения информации об ошибке необходимо вызвать функцию GetLastError().

См. также ObjectGetValueByShift().

**Параметры:**

**name** - Имя объекта.

**value** - Значение цены.

**Пример:**

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

## 16.8 ObjectGetValueByShift

```
double ObjectGetValueByShift(string name, int shift)
```

Функция вычисляет и возвращает значение цены для указанного бара (смещение относительно текущего бара). Значение цены вычисляется при помощи линейного уравнения по первой и второй координатам. Применяется для трендовых линий и аналогичных объектов. Для получения информации об ошибке необходимо вызвать функцию GetLastError().

См. также ObjectGetShiftByValue().

### Параметры:

**name** - Имя объекта.  
**shift** - Номер бара.

### Пример:

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

## 16.9 ObjectMove

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

Изменение одной из координат объекта на графике. Объекты могут иметь от одной до трех точек привязки в зависимости от типа объекта. Функция возвращает TRUE в случае успеха, иначе FALSE. Для получения дополнительной информации об ошибке необходимо вызвать функцию GetLastError().

Нумерация координат объекта начинается с 0.

### Параметры:

**name** - Имя объекта.  
**point** - Индекс координаты (0-2).  
**time1** - Новое значение времени.  
**price1** - Новое значение цены.

### Пример:

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

## 16.10 ObjectName

`string ObjectName(int index)`

Функция возвращает имя объекта по порядковому номеру в списке объектов. Для получения дополнительной информации об ошибке необходимо вызвать функцию `GetLastError()`.

### Параметры:

**index** - Порядковый номер в списке объектов. Должен быть большим или равным 0 и меньшим, чем `ObjectsTotal()`.

### Пример:

```
int    obj_total=ObjectsTotal();
string name;
for(int i=0;i<obj_total;i++)
{
    name=ObjectName(i);
    Print(i,": Имя объекта - ",name);
}
```

## 16.11 ObjectsDeleteAll

`int ObjectsDeleteAll(int window=EMPTY, int type=EMPTY)`

Удаление всех объектов с указанным типом и в указанном подокне графика. Функция возвращает число удаленных объектов. Для получения дополнительной информации об ошибке необходимо вызвать функцию `GetLastError()`.

Замечания: нумерация подокон графика (если на графике есть подокна с индикаторами) начинается с 1. Главное окно графика есть всегда и имеет индекс 0. Если индекс окна отсутствует или имеет значение -1, то объекты удаляются со всего графика.

Если значение параметра *type* равно -1 или этот параметр отсутствует, то удаляются все объекты из указанного подокна.

### Параметры:

**window** - Необязательный параметр. Индекс окна, на котором будут удалены объекты. Должен быть большим или равным -1 (EMPTY, значение по умолчанию) и меньшим, чем `WindowsTotal()`.

**type** - Необязательный параметр. Тип объекта для удаления. Это может быть любое из значений списка идентификаторов типов объектов или EMPTY (-1) для удаления всех объектов.

### Пример:

```

ObjectsDeleteAll(2, OBJ_HLINE); // удаляются все горизонтальные линии из
2-го подокна.
ObjectsDeleteAll(2);           // удаляются все объекты из 2-го подокна.
ObjectsDeleteAll();           // удаляются все объекты с графика.

```

## 16.12 ObjectSet

**bool ObjectSet(string name, int prop\_id, double value)**

Изменение значения указанного свойства объекта. В случае успеха функция возвращает TRUE, иначе FALSE. Для получения информации об ошибке необходимо вызвать функцию GetLastError().

См. также ObjectGet().

**Параметры:**

**name** - Имя объекта.

**prop\_id** - Идентификатор свойства объекта. Может быть любым из списка свойств объекта.

**value** - Новое значение указанного свойства.

**Пример:**

```

// перемещение первой координаты в конец графика
ObjectSet("MyTrend", OBJPROP_TIME1, Time[0]);
// изменение значения второго уровня объекта fibo
ObjectSet("MyFibo", OBJPROP_FIRSTLEVEL+1, 1.234);
// установка флага видимости объекта. Объект будет отрисован только на
15-минутном и 1-часовом периодах графика
ObjectSet("MyObject", OBJPROP_TIMEFRAMES, OBJ_PERIOD_M15 |
OBJ_PERIOD_H1);

```

## 16.13 ObjectSetFiboDescription

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

Функция присваивает новое описание уровню объекта Фибоначчи. Количество уровней зависит от типа объекта Фибоначчи. Максимальное количество уровней - 32.

Для получения информации об ошибке необходимо вызвать функцию GetLastError().

**Параметры:**

**name** - Имя объекта.

**index** - Порядковый номер уровня объекта Фибоначчи (0-31).

**text** - Новое описание уровня.

Пример:

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

### 16.14 ObjectSetText

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

Изменение описания объекта. Для объектов OBJ\_TEXT и OBJ\_LABEL это описание отображается на графике в виде текстовой строки. В случае успеха функция возвращает значение TRUE, иначе FALSE. Для получения дополнительной информации об ошибке необходимо вызвать функцию GetLastError().

Параметры *font\_size*, *font\_name* и *text\_color* используются только для объектов OBJ\_TEXT и OBJ\_LABEL. Для объектов других типов эти параметры игнорируются.

См. также ObjectDescription().

Параметры:

**name** - Имя объекта.  
**text** - Текст описания объекта.  
**font\_size** - Размер шрифта в пунктах.  
**font\_name** - Наименование шрифта.  
**text\_color** - Цвет текста.

Пример:

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

← ObjectSetFiboDescription ObjectsTotal →

### 16.15 ObjectsTotal

```
int ObjectsTotal(int type=EMPTY)
```

Возвращает общее число объектов указанного типа на графике.

Параметры:

**type** - Необязательный параметр. Тип объекта для подсчета количества объектов данного типа. Это может быть любое из значений списка идентификаторов типов объектов или EMPTY (-1) для подсчета всех

объектов.

**Пример:**

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

### **16.16 ObjectType**

**int ObjectType(string name)**

Функция возвращает тип указанного объекта. Для получения информации об ошибке необходимо вызвать функцию GetLastError().

**Параметры:**

**name** - Имя объекта.

**Пример:**

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

## 17 Строковые функции

Группа функций, предназначенных для работы с данными типа `string`.

### 17.1 StringConcatenate

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

Формирует строку из переданных параметров и возвращает её. Параметры могут иметь любой тип. Количество параметров не может превышать 64.

Параметры преобразуются в строки по тем же правилам, что и в функциях `Print()`, `Alert()` и `Comment()`. Возвращаемая строка получается в результате конкатенации строк, преобразованных из параметров функции.

Функция `StringConcatenate()` работает быстрее и экономнее по памяти, чем связывание строк при помощи операций сложения (+).

#### Параметры:

... - Любые значения, разделенные запятыми.

#### Пример:

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

### 17.2 StringFind

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

Поиск подстроки. Возвращает номер позиции в строке, с которой начинается искомая подстрока, либо -1, если подстрока не найдена.

#### Параметры:

**text** - Строка, в которой производится поиск.  
**matched\_text** - Искомая подстрока.  
**start** - Позиция в строке, с которой должен быть начат поиск.

#### Пример:

```
string text = "Быстрая коричневая собака перепрыгивает ленивую лисицу";
int index=StringFind(text, "собака перепрыгивает", 0);
```

```
if(index!=20)
    Print("oops!");
```

### 17.3 StringGetChar

**int StringGetChar(string text, int pos)**

Возвращает значение символа, расположенного в указанной позиции строки.

**Параметры:**

**text** - Строка.

**pos** - Позиция символа в строке. Может быть от 0 до StringLen(text)-1.

**Пример:**

```
int char_code=StringGetChar("abcdefgh", 3);
// СИМВОЛЬНЫЙ КОД 'c' = 99
```

### 17.4 StringLen

**int StringLen(string text)**

Возвращает число символов в строке.

**Параметры:**

**text** - Строка для вычисления длины.

**Пример:**

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

### 17.5 StringSetChar

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

Возвращает копию строки с измененным значением символа в указанной позиции.

**Параметры:**

**text** - Строка для изменения.

**pos** - Позиция символа в строке. Может быть от 0 до StringLen(text).

**value** - Символьный код ASCII.

**Пример:**

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

## 17.6 StringSubstr

```
string StringSubstr(string text, int start, int length=0)
```

Извлекает подстроку из текстовой строки, начинающейся с указанной позиции.

Функция возвращает копию извлеченной подстроки, если возможно, иначе возвращается пустая строка.

**Параметры:**

- text** - Строка, из которой должна быть извлечена подстрока.
- start** - Начальная позиция подстроки. Может быть от 0 до `StringLen(text)-1`.
- length** - Длина извлекаемой подстроки. Если значение параметра меньше или равно 0 либо параметр не задан, то будет извлекаться подстрока, начиная с указанной позиции и до конца строки.

**Пример:**

```
string text = "Быстрая коричневая собака перепрыгивает ленивую лисицу";  
string substr = StringSubstr(text, 8, 10);  
// извлеченная строка "коричневая"
```

## 17.7 StringTrimLeft

```
string StringTrimLeft(string text)
```

Функция урезает символы перевода каретки, пробелы и символы табуляции в левой части строки. Функция возвращает копию преобразованной строки, если это возможно, в противном случае возвращается пустая строка.

**Параметры:**

- text** - Строка, которая будет срезана слева.

**Пример:**

```
string str1=" Hello world ";  
string str2=StringTrimLeft(str);  
// после срезания str2, у переменной будет значение - "Hello World "
```

## 17.8 StringTrimRight

```
string StringTrimRight(string text)
```

Функция урезает символы перевода каретки, пробелы и символы табуляции в правой части строки. Функция возвращает копию преобразованной строки, если это возможно, в противном случае возвращается пустая строка.

**Параметры:**

**text** - Строка, которая будет срезана справа.

**Пример:**

```
string str1 = " Hello world ";  
string str2=StringTrimRight(str);  
// после срезания str2, у переменной будет значение - " Hello World"
```

## 18 Технические индикаторы

Группа функций, предназначенных для расчета стандартных и пользовательских индикаторов.

Для того, чтобы эксперт (или любая MQL4-программа) мог получить значение какого-либо индикатора, присутствие данного индикатора на текущем графике необязательно. Запрошенный индикатор будет загружен и рассчитан в потоке вызвавшего его модуля.

Любой индикатор может быть рассчитан на данных не только текущего графика, но и на данных любого доступного символа/периода. Если запрашивается информация с другого графика (название инструмента и/или значение таймфрейма отличаются от текущих), то возможна ситуация, что в клиентском терминале не открыт соответствующий график и необходимые данные должны быть запрошены у сервера. В этом случае в переменную `last_error` будет помещена ошибка ERR\_HISTORY\_WILL\_UPDATED (4066 - запрошенные исторические данные в состоянии обновления) и необходимо через некоторое время повторить попытку запроса (см. пример `ArrayCopySeries()`).

### 18.1 iAC

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

Расчет осциллятора Accelerator/Decelerator.

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

## 18.2 *iAD*

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

Расчет индикатора Accumulation/Distribution.

Параметры:

- |                  |  |
|------------------|--|
| <b>symbol</b>    | - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.                      |
| <b>timeframe</b> | - Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.                                      |
| <b>shift</b>     | - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад). |

Пример:

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

## 18.3 *iAlligator*

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

Расчет индикатора Alligator.

Параметры:

- |                     |   |
|---------------------|---|
| <b>symbol</b>       | - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ. |
| <b>timeframe</b>    | - Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.                 |
| <b>jaw_period</b>   | - Период усреднения синей линии (челюсти аллигатора).   |
| <b>jaw_shift</b>    | - Смещение синей линии относительно графика цены.   |
| <b>teeth_period</b> | - Период усреднения красной линии (зубов аллигатора).   |
| <b>teeth_shift</b>  | - Смещение красной линии относительно графика цены.   |
| <b>lips_period</b>  | - Период усреднения зеленой линии (губ аллигатора).   |
| <b>lips_shift</b>   | - Смещение зеленой линии относительно графика цены.   |

<b>ma_method</b>	- Метод усреднения. Может быть любым из значений <u>методов скользящего среднего (Moving Average)</u> .
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых КОНСТАНТ</u> .
<b>mode</b>	- Источник данных, идентификатор одной из линий индикатора. Может быть любой из следующих величин: MODE_GATORJAW - синяя линия (линия челюсти аллигатора), MODE_GATORTEETH - красная линия (линия зубов аллигатора), MODE_GATORLIPS - зеленая линия (линия губ аллигатора).
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

#### Пример:

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

### 18.4 *iADX*

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

Расчет Average Directional Movement Index.

#### Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>period</b>	- Период усреднения для вычисления индекса.
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых КОНСТАНТ</u> .
<b>mode</b>	- Индекс линии индикатора. Может быть любым из

перечисленных идентификаторов линии индикаторов.

**shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

## 18.5 iATR

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

Расчет индикатора Average True Range.

**Параметры:**

**symbol** - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.

**timeframe** - Период. Может быть одним из периодов графика. 0 означает период текущего графика.

**period** - Период усреднения для вычисления индикатора.

**shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

## 18.6 iAO

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

Расчет Awesome oscillator.

**Параметры:**

**symbol** - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.

**timeframe** - Период. Может быть одним из периодов графика. 0 означает период текущего графика.

**shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

## 18.7 iBearsPower

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

Расчет индикатора Bears Power.

Параметры:

<b>symbol</b>	-	Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	-	Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>period</b>	-	Период усреднения для вычисления индикатора.
<b>applied_price</b>	-	Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>shift</b>	-	Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

## 18.8 iBands

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

Расчет индикатора Bollinger Bands.

Параметры:

<b>symbol</b>	-	Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	-	Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>period</b>	-	Период усреднения основной линии индикатора.
<b>deviation</b>	-	Отклонение от основной линии.
<b>bands_shift</b>	-	Сдвиг индикатора относительно ценового графика.
<b>applied_price</b>	-	Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>mode</b>	-	Индекс линии индикатора. Может быть любым из <u>идентификаторов линий индикаторов</u> .
<b>shift</b>	-	Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

### 18.9 *iBandsOnArray*

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

Расчет индикатора Bollinger Bands на данных, хранящихся в массиве. В отличие от `iBands(...)` функция `iBandsOnArray` не выбирает данные на основе названия инструмента, таймфрейма и используемой цены - ценовые данные должны быть подготовлены заранее. Расчет производится слева направо. Для организации доступа к элементам массива, как к таймсерии (то есть справа налево), необходимо использовать функцию [ArraySetAsSeries](#).

Параметры:

<b>array[]</b>	- Массив с данными.
<b>total</b>	- Количество элементов для вычисления. 0 означает все элементы массива.
<b>period</b>	- Период усреднения основной линии индикатора.
<b>deviation</b>	- Отклонение от основной линии.
<b>bands_shift</b>	- Сдвиг индикатора относительно ценового графика.
<b>mode</b>	- Индекс линии индикатора. Может быть любым из <a href="#">идентификаторов линий индикаторов</a> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

### 18.10 *iBullsPower*

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

Расчет индикатора Bulls Power.

Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>period</b>	- Период усреднения для вычисления индикатора.
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### 18.11 iCCI

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

Расчет индикатора Commodity Channel Index.

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>period</b>	- Период усреднения для вычисления индикатора.
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

## 18.12 iCCIOnArray

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

Расчет индикатора Commodity Channel Index на данных, хранящихся в массиве. В отличие от iCCI(...) функция iCCIOnArray не выбирает данные на основе названия инструмента, таймфрейма и используемой цены - ценовые данные должны быть подготовлены заранее. Расчет производится слева направо. Для организации доступа к элементам массива, как к таймсерии (то есть справа налево), необходимо использовать функцию [ArraySetAsSeries](#).

### Параметры:

- array[]** - Массив с данными.
- total** - Количество элементов для вычисления. 0 означает все элементы массива.
- period** - Период усреднения для вычисления индикатора.
- shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

### Пример:

```
if(iCCIOnArray(ExtBuffer,total,12,0)>iCCIOnArray(ExtBuffer,total,20,0))  
return(1);
```

## 18.13 iCustom

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

Расчет указанного пользовательского индикатора. Пользовательский индикатор должен быть скомпилирован (файл с расширением EX4) и находиться в директории *каталог\_терминала\experts\indicators*.

### Параметры:

- symbol** - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из периодов графика. 0 означает период текущего графика.

<b>name</b>	- Имя пользовательского индикатора.
<b>...</b>	- Список параметров (при необходимости). Передаваемые параметры должны соответствовать порядку объявления и типу внешних (extern) переменных пользовательского индикатора.
<b>mode</b>	- Индекс линии индикатора. Может быть от 0 до 7 и должен соответствовать индексу, используемому одной из функций <u>SetIndexBuffer</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.14 iDeMarker**

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

Расчет индикатора DeMarker.

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>period</b>	- Период усреднения для вычисления индикатора.
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.15 iEnvelopes**

```
double          string symbol, int timeframe, int ma_period,
```

```
iEnvelopes( int ma_method, int ma_shift, int applied_price,  
            double deviation, int mode, int shift)
```

Расчет индикатора Envelopes.

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>ma_period</b>	- Период усреднения основной линии индикатора.
<b>ma_method</b>	- Метод усреднения. Может быть любым из значений <u>методов скользящего среднего (Moving Average)</u> .
<b>ma_shift</b>	- Сдвиг индикатора относительно ценового графика.
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>deviation</b>	- Отклонение от основной линии в процентах.
<b>mode</b>	- Индекс линии индикатора. Может быть любым из значений <u>идентификаторов линий индикаторов</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.16 iEnvelopesOnArray**

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

Расчет индикатора Envelopes на данных, хранящихся в массиве. В отличие от iEnvelopes(...) функция iEnvelopesOnArray не выбирает данные на основе названия инструмента, таймфрейма и используемой цены - ценовые данные должны быть подготовлены заранее. Расчет производится слева направо. Для организации доступа к

элементам массива, как к таймсерии (то есть справа налево), необходимо использовать функцию `ArraySetAsSeries`.

#### Параметры:

<b>array[]</b>	- Массив с данными.
<b>total</b>	- Количество элементов для вычисления. 0 означает все элементы массива.
<b>ma_period</b>	- Период усреднения основной линии индикатора.
<b>ma_method</b>	- Метод усреднения. Может быть любым из значений <u>методов скользящей средней (Moving Average)</u> .
<b>ma_shift</b>	- Сдвиг индикатора относительно ценового графика.
<b>deviation</b>	- Отклонение от основной линии в процентах.
<b>mode</b>	- Индекс линии индикатора. Может быть любым из значений <u>идентификаторов линий индикаторов</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

#### Пример:

```
double  
val=iEnvelopesOnArray(ExtBuffer,10,13,MODE_SMA,0,0.2,MODE_UPPER,0);
```

### **18.17 iForce**

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

Расчет Force Index.

#### Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>period</b>	- Период усреднения для вычисления индикатора.
<b>ma_method</b>	- Метод усреднения. Может быть любым из значений <u>методов</u>

скользящей средней (Moving Average).

- applied\_price** - Используемая цена. Может быть любой из ценовых КОНСТАНТ.
- shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

### 18.18 iFractals

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

Расчет индикатора Fractals.

Параметры:

- symbol** - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из периодов графика. 0 означает период текущего графика.
- mode** - Индекс линии индикатора. Может быть любым из значений идентификаторов линии индикаторов.
- shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

```
double val=iFractals(NULL, 0, MODE_UPPER, 3);
```

### 18.19 iGator

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

Расчет осциллятора Gator. Осциллятор показывает разницу между синей и красной линией Аллигатора (верхняя гистограмма) и разницу между красной и зеленой линией (нижняя гистограмма).

#### Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>jaw_period</b>	- Период усреднения синей линии (челюсти аллигатора).
<b>jaw_shift</b>	- Смещение синей линии относительно графика цены.
<b>teeth_period</b>	- Период усреднения красной линии (зубов аллигатора).
<b>teeth_shift</b>	- Смещение красной линии относительно графика цены.
<b>lips_period</b>	- Период усреднения зеленой линии (губ аллигатора).
<b>lips_shift</b>	- Смещение зеленой линии относительно графика цены.
<b>ma_method</b>	- Метод усреднения. Может быть любым из значений <u>методов скользящей средней (Moving Average)</u> .
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>mode</b>	- Индекс линии индикатора. Может быть любым из значений <u>идентификаторов линии индикаторов</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

#### Пример:

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

### **18.20 ilchimoku**

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

Расчет индикатора Ichimoku Kinko Hyo.

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>tenkan_sen</b>	- Период усреднения Tenkan Sen.
<b>kijun_sen</b>	- Период усреднения Kijun Sen.
<b>senkou_span_b</b>	- Период усреднения Senkou Span B.
<b>mode</b>	- Источник данных. Может быть одним из перечисленных <u>идентификаторов Ichimoku Kinko Hyo</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.21 iBWMFI**

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

Расчет Market Facilitation Index.

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

## 18.22 *iMomentum*

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

Расчет индикатора Momentum.

Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>period</b>	- Период(количество баров) для вычисления изменения цены.
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых КОНСТАНТ</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

## 18.23 *iMomentumOnArray*

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

Расчет индикатора Momentum на данных, хранящихся в массиве. В отличие от iMomentum(...) функция iMomentumOnArray не выбирает данные на основе названия инструмента, таймфрейма и используемой цены - ценовые данные должны быть подготовлены заранее. Расчет производится слева направо. Для организации доступа к элементам массива, как к таймсерии (то есть справа налево), необходимо использовать функцию ArraySetAsSeries.

Параметры:

<b>array[]</b>	- Массив с данными.
<b>total</b>	- Количество элементов для вычисления. 0 означает все элементы массива.

- period** - Период(количество баров) для вычисления изменения цены.
- shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

### 18.24 *iMFI*

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

Расчет Money Flow Index.

Параметры:

- symbol** - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из периодов графика. 0 означает период текущего графика.
- period** - Период(количество баров) для вычисления индикатора.
- shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

### 18.25 *iMA*

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

Расчет скользящего среднего.

Параметры:

- symbol** - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из периодов графика. 0 означает

	период текущего графика.
<b>period</b>	- Период усреднения для вычисления скользящего среднего.
<b>ma_shift</b>	- Сдвиг индикатора относительно ценового графика.
<b>ma_method</b>	- Метод усреднения. Может быть любым из значений <u>методов скользящего среднего (Moving Average)</u> .
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.26 iMAOnArray**

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

Расчет скользящего среднего на данных, хранящихся в массиве. В отличие от iMA(...) функция iMAOnArray не выбирает данные на основе названия инструмента, таймфрейма и используемой цены - ценовые данные должны быть подготовлены заранее. Расчет производится слева направо. Для организации доступа к элементам массива, как к таймсерии (то есть справа налево), необходимо использовать функцию [ArraySetAsSeries](#).

**Параметры:**

<b>array[]</b>	- Массив с данными.
<b>total</b>	- Количество элементов для вычисления. 0 означает все элементы массива.
<b>period</b>	- Период усреднения для вычисления скользящего среднего.
<b>ma_shift</b>	- Сдвиг индикатора относительно ценового графика.
<b>ma_method</b>	- Метод усреднения. Может быть любым из значений <u>методов скользящего среднего (Moving Average)</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

## Пример:

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

## 18.27 iOsMA

<code>double</code>	<code>string symbol, int timeframe, int fast_ema_period,</code>
<code>iOsMA(</code>	<code>int slow_ema_period, int signal_period,</code>
	<code>int applied_price, int shift)</code>

Расчет Moving Average of Oscillator. Осциллятор OsMA показывает разницу между значениями MACD и его сигнальной линии. В некоторых системах этот осциллятор называется гистограммой MACD.

### Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>fast_ema_period</b>	- Период усреднения для вычисления быстрой скользящей средней.
<b>slow_ema_period</b>	- Период усреднения для вычисления медленной скользящей средней.
<b>signal_period</b>	- Период усреднения для вычисления сигнальной линии.
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ЦЕНОВЫХ КОНСТАНТ</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

### Пример:

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

## 18.28 iMACD

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

Расчет индикатора Moving Averages Convergence/Divergence. В тех системах, где OsMA называют гистограммой МАКД, данный индикатор изображается в виде двух линий. В клиентском терминале схождение/расхождение скользящих средних рисуется в виде гистограммы.

### Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>fast_ema_period</b>	- Период усреднения для вычисления быстрой скользящей средней.
<b>slow_ema_period</b>	- Период усреднения для вычисления медленной скользящей средней.
<b>signal_period</b>	- Период усреднения для вычисления сигнальной линии.
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>mode</b>	- Индекс линии индикатора. Может быть любым из значений <u>идентификаторов линии индикаторов</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

### Пример:

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

### 18.29 *iOBV*

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

Расчет индикатора On Balance Volume.

Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

### 18.30 *iSAR*

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

Расчет индикатора Parabolic Stop and Reverse system.

Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>step</b>	- Приращение уровня стопа, обычно 0.02.
<b>maximum</b>	- Максимальный уровень стопа, обычно 0.2.
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

### 18.31 *iRSI*

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

Расчет Relative Strength Index.

Параметры:

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>period</b>	- Период усреднения для вычисления индекса.
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых КОНСТАНТ</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

Пример:

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

### 18.32 *iRSIOnArray*

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

Расчет индикатора Relative Strength Index на данных, хранящихся в массиве. В отличие от iRSI(...) функция iRSI не выбирает данные на основе названия инструмента, таймфрейма и используемой цены - ценовые данные должны быть подготовлены заранее. Расчет производится слева направо. Для организации доступа к элементам массива, как к таймсерии (то есть справа налево), необходимо использовать функцию ArraySetAsSeries.

Параметры:

<b>array[]</b>	- Массив с данными.
----------------	---------------------

- total** - Количество элементов для вычисления. 0 означает все элементы массива.
- period** - Период усреднения для вычисления индекса.
- shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.33 iRVI**

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

Расчет Relative Vigor Index.

**Параметры:**

- symbol** - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из периодов графика. 0 означает период текущего графика.
- period** - Период усреднения для вычисления индекса.
- mode** - Индекс линии индикатора. Может быть любым из значений идентификаторов линии индикаторов.
- shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.34 iStdDev**

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

`int shift)`

Расчет индикатора Standard Deviation.

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>ma_period</b>	- Период усреднения для вычисления индикатора.
<b>ma_shift</b>	- Сдвиг индикатора относительно ценового графика.
<b>ma_method</b>	- Метод усреднения. Может быть любым из значений <u>методов скользящего среднего (Moving Average)</u> .
<b>applied_price</b>	- Используемая цена. Может быть любой из <u>ценовых констант</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.35 iStdDevOnArray**

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

Расчет индикатора Standard Deviation на данных, хранящихся в массиве. В отличие от `iStdDev(...)` функция `iStdDevOnArray` не выбирает данные на основе названия инструмента, таймфрейма и используемой цены - ценовые данные должны быть подготовлены заранее. Расчет производится слева направо. Для организации доступа к элементам массива, как к таймсерии (то есть справа налево), необходимо использовать функцию `ArraySetAsSeries`.

**Параметры:**

<b>array[]</b>	- Массив с данными.
<b>total</b>	- Количество элементов для вычисления. 0 означает все

	элементы массива.
<b>ma_period</b>	- Период усреднения для вычисления индикатора.
<b>ma_shift</b>	- Сдвиг индикатора относительно ценового графика.
<b>ma_method</b>	- Метод усреднения. Может быть любым из значений <u>методов скользящего среднего (Moving Average)</u> .
<b>shift</b>	- Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.36 iStochastic**

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

Расчет Stochastic Oscillator.

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>%Kperiod</b>	- Период(количество баров) для вычисления линии %K.
<b>%Dperiod</b>	- Период усреднения для вычисления линии %D.
<b>slowing</b>	- Значение замедления.
<b>method</b>	- Метод усреднения. Может быть любым из значений <u>методов скользящего среднего (Moving Average)</u> .
<b>price_field</b>	- Параметр выбора цен для расчета. Может быть одной из следующих величин: 0 - Low/High или 1 - Close/Close.
<b>mode</b>	- Индекс линии индикатора. Может быть любым из значений <u>идентификаторов линий индикаторов</u> .

**shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **18.37 iWPR**

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

Расчет индикатора Larry Williams' Percent Range.

**Параметры:**

**symbol** - Символьное имя инструмента, на данных которого будет вычисляться индикатор. NULL означает текущий символ.

**timeframe** - Период. Может быть одним из периодов графика. 0 означает период текущего графика.

**period** - Период(количество баров) для вычисления индикатора.

**shift** - Индекс получаемого значения из индикаторного буфера (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

## 19 Доступ к таймсериям

Группа функций, предназначенных для доступа к ценовым данным любого доступного символа/периода.

Если запрашивается информация с другого графика (название инструмента и/или значение таймфрейма отличаются от текущих), то возможна ситуация, что в клиентском терминале не открыт соответствующий график и необходимые данные должны быть запрошены у сервера. В этом случае в переменную `last_error` будет помещена ошибка `ERR_HISTORY_WILL_UPDATED` (4066 - запрошенные исторические данные в состоянии обновления) и необходимо через некоторое время повторить попытку запроса (см. пример `ArrayCopySeries()`).

При тестировании ценовые данные того же самого символа, но другого таймфрейма, моделируются точно, за исключением объемов. Объемы других таймфреймов не моделируются. Ценовые данные других символов не моделируются. Во всех случаях количество баров в таймсериях моделируется точно

### 19.1 *iBars*

```
int iBars(string symbol, int timeframe)
```

Возвращает количество баров на определенном графике.

Для текущего графика информация о количестве баров находится в предопределенной переменной `Bars`.

#### Параметры:

- symbol** - Символьное имя инструмента. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из периодов графика. 0 означает период текущего графика.

#### Пример:

```
Print("Bar count on the 'EURUSD,H1' is ",iBars("EURUSD",PERIOD_H1));
```

### 19.2 *iBarShift*

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

Поиск бара по времени. Функция возвращает смещение бара, которому принадлежит указанное время. Если для указанного времени бар отсутствует ("дыра" в истории), то функция возвращает, в зависимости от параметра *exact*, -1 или смещение ближайшего бара.

#### Параметры:

<b>symbol</b>	- Символьное имя инструмента. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>time</b>	- Значение времени для поиска.
<b>exact</b>	- Возвращаемое значение если бар не найден. FALSE - iBarShift возвращает ближайший. TRUE - iBarShift возвращает -1.

#### Пример:

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

### 19.3 iClose

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

Возвращает значение цены закрытия указанного параметром *shift* бара с соответствующего графика (*symbol*, *timeframe*). В случае ошибки функция возвращает 0. Для получения дополнительной информации об ошибке необходимо вызвать функцию GetLastError().

Для текущего графика информация о ценах закрытия находится в предопределенном массиве Close[].

#### Параметры:

<b>symbol</b>	- Символьное имя инструмента. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>shift</b>	- Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество периодов назад).

#### Пример:

```

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

```

## 19.4 *iHigh*

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

Возвращает значение максимальной цены указанного параметром *shift* бара с соответствующего графика (*symbol*, *timeframe*). В случае ошибки функция возвращает 0. Для получения дополнительной информации об ошибке необходимо вызвать функцию [GetLastError\(\)](#).

Для текущего графика информация о максимальных ценах находится в предопределенном массиве [High\[\]](#).

**Параметры:**

- symbol** - Символьное имя инструмента. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из [периодов графика](#). 0 означает период текущего графика.
- shift** - Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество периодов назад).

**Пример:**

```

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

```

## 19.5 *iHighest*

**int iHighest(string symbol, int timeframe, int type, int count=WHOLE\_ARRAY, int start=0)**

Возвращает индекс найденного наибольшего значения (смещение относительно текущего бара).

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента, на данных которого будет производиться поиск. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>type</b>	- Идентификатор таймсерии. Может быть любым из значений <u>идентификаторов таймсерий</u> .
<b>count</b>	- Число элементов таймсерии (в направлении от текущего бара в сторону возрастания индекса), среди которых должен быть произведен поиск.
<b>start</b>	- Индекс (смещение относительно текущего бара) начального бара, с которого начинается поиск наибольшего значения. Отрицательные значения игнорируются и заменяются нулевым значением.

**Пример:**

```
double val;
// расчет максимального значения цены на 20 последовательных барах
// с индекса 4 по индекс 23 включительно на текущем графике
val=High[iHighest(NULL,0,MODE_HIGH,20,4)];
```

## 19.6 iLow

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

Возвращает значение минимальной цены указанного параметром *shift* бара с соответствующего графика (*symbol*, *timeframe*). В случае ошибки функция возвращает 0.

Для получения дополнительной информации об ошибке необходимо вызвать функцию GetLastError().

Для текущего графика информация о минимальных ценах находится в предопределенном массиве Low[].

**Параметры:**

<b>symbol</b>	- Символьное имя инструмента. NULL означает текущий символ.
<b>timeframe</b>	- Период. Может быть одним из <u>периодов графика</u> . 0 означает период текущего графика.
<b>shift</b>	- Индекс получаемого значения из таймсерии (сдвиг

относительно текущего бара на указанное количество периодов назад).

**Пример:**

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

### **19.7 iLowest**

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

Возвращает индекс найденного наименьшего значения (смещение относительно текущего бара).

**Параметры:**

- symbol** - Символьное имя инструмента, на данных которого будет производиться поиск. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из значений периодов графика. 0 означает период текущего графика.
- type** - Идентификатор таймсерии. Может быть любым из значений идентификаторов таймсерий.
- count** - Число элементов таймсерии (в направлении от текущего бара в сторону возрастания индекса), среди которых должен быть произведен поиск.
- start** - Смещение (относительно текущего) начального бара, с которого начинается поиск наименьшего значения.

**Пример:**

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

### **19.8 iOpen**

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

Возвращает значение цены открытия указанного параметром *shift* бара с соответствующего графика (*symbol*, *timeframe*). В случае ошибки функция возвращает 0.

Для получения дополнительной информации об ошибке необходимо вызвать функцию [GetLastError\(\)](#).

Для текущего графика информация о ценах открытия находится в предопределенном массиве [Open\[\]](#).

#### Параметры:

- symbol** - Символьное имя инструмента. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из [периодов графика](#). 0 означает период текущего графика.
- shift** - Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество периодов назад).

#### Пример:

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

## 19.9 [iTime](#)

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

Возвращает значение времени открытия указанного параметром *shift* бара с соответствующего графика (*symbol*, *timeframe*). В случае ошибки функция возвращает 0.

Для получения дополнительной информации об ошибке необходимо вызвать функцию [GetLastError\(\)](#).

Для текущего графика информация о времени открытия каждого бара находится в предопределенном массиве [Time\[\]](#).

#### Параметры:

- symbol** - Символьное имя инструмента. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из [периодов графика](#). 0 означает период текущего графика.
- shift** - Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество периодов назад).

### Пример:

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

## 19.10 *iVolume*

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

Возвращает значение тикового объема указанного параметром *shift* бара с соответствующего графика (*symbol*, *timeframe*). В случае ошибки функция возвращает 0. Для получения дополнительной информации об ошибке необходимо вызвать функцию [GetLastError\(\)](#).

Для текущего графика информация о тиковых объемах каждого бара находится в определенном массиве [Volume\[\]](#).

### Параметры:

- symbol** - Символьное имя инструмента. NULL означает текущий символ.
- timeframe** - Период. Может быть одним из [периодов графика](#). 0 означает период текущего графика.
- shift** - Индекс получаемого значения из таймсерии (сдвиг относительно текущего бара на указанное количество периодов назад).

### Пример:

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

## **20 Торговые функции**

Группа функций, предназначенных для управления торговой деятельностью.

Торговые функции OrderSend(), OrderClose, OrderCloseBy, OrderDelete и OrderModify не могут быть вызваны из пользовательских индикаторов.

Торговые функции могут использоваться в экспертах и скриптах. Торговые функции могут быть вызваны только в том случае, если в свойствах соответствующего эксперта или скрипта включена галочка "Разрешить советнику торговать".

Для проведения торговых операций из экспертов и скриптов предусмотрен всего один поток, который запускается в программном торговом контексте (контекст автоматической торговли из экспертов и скриптов). Поэтому, если этот контекст занят торговой операцией какого-либо эксперта, то другой эксперт или скрипт не может в этот момент вызывать торговые функции из-за ошибки 146 (ERR\_TRADE\_CONTEXT\_BUSY). Для определения возможности выполнять торговые операции необходимо использовать функцию IsTradeAllowed(). Для чёткого разделения доступа к торговому контексту можно использовать семафор на основе глобальной переменной, значение которой необходимо менять при помощи функции GlobalVariableSetOnCondition().

### **20.1 Ошибки исполнения**

Любая торговая операция (функции OrderSend(), OrderClose, OrderCloseBy, OrderDelete или OrderModify) по ряду причин может завершиться неудачей и вернуть либо отрицательный номер тикета, либо FALSE. Причину неудачи можно выяснить, вызвав функцию GetLastError(). Каждая ошибка должна быть обработана по-своему. Ниже в таблице приведены общие рекомендации.

Коды ошибок, возвращаемые торговым сервером:

<b>Константа</b>	<b>Код</b>	<b>Описание</b>
ERR_NO_ERROR	0	Торговая операция прошла успешно.
ERR_NO_RESULT	1	<u>OrderModify</u> пытается изменить

		уже установленные значения такими же значениями. Необходимо изменить одно или несколько значений и повторить попытку.
ERR_COMMON_ERROR	2	Общая ошибка. Прекратить все попытки торговых операций до выяснения обстоятельств. Возможно перезагрузить операционную систему и клиентский терминал.
ERR_INVALID_TRADE_PARAMETERS	3	В торговую функцию переданы неправильные параметры, например, неправильный символ, неопознанная <u>торговая операция</u> , отрицательное допустимое отклонение цены, несуществующий номер тикета и т.п. Необходимо изменить логику программы.
ERR_SERVER_BUSY	4	Торговый сервер занят. Можно повторить попытку через достаточно большой промежуток времени (от нескольких минут).
ERR_OLD_VERSION	5	Старая версия клиентского терминала. Необходимо установить последнюю версию клиентского терминала.
ERR_NO_CONNECTION	6	Нет связи с торговым сервером. Необходимо убедиться, что

		связь не нарушена (например, при помощи функции <u>IsConnected</u> ) и через небольшой промежуток времени (от 5 секунд) повторить попытку.
ERR_TOO_FREQUENT_REQUESTS	8	Слишком частые запросы. Необходимо уменьшить частоту запросов, изменить логику программы.
ERR_ACCOUNT_DISABLED	64	Счет заблокирован. Необходимо прекратить все попытки торговых операций.
ERR_INVALID_ACCOUNT	65	Неправильный номер счета. Необходимо прекратить все попытки торговых операций.
ERR_TRADE_TIMEOUT	128	Истек срок ожидания совершения сделки. Прежде, чем производить повторную попытку (не менее, чем через 1 минуту), необходимо убедиться, что торговая операция действительно не прошла (новая позиция не была открыта, либо существующий ордер не был изменён или удалён, либо существующая позиция не была закрыта)
ERR_INVALID_PRICE	129	Неправильная цена bid или ask, возможно, ненормализованная цена. Необходимо после задержки от 5 секунд обновить

		<p><u>данные</u> при помощи функции <u>RefreshRates</u> и повторить попытку. Если ошибка не исчезает, необходимо прекратить все попытки торговых операций и изменить логику программы.</p>
ERR_INVALID_STOPS	130	<p>Слишком близкие стопы или неправильно рассчитанные или ненормализованные цены в стопах (или в цене открытия отложенного ордера). Попытку можно повторять только в том случае, если ошибка произошла из-за устаревания цены.</p> <p>Необходимо после задержки от 5 секунд обновить <u>данные</u> при помощи функции <u>RefreshRates</u> и повторить попытку. Если ошибка не исчезает, необходимо прекратить все попытки торговых операций и изменить логику программы.</p>
ERR_INVALID_TRADE_VOLUME	131	<p>Неправильный объем, ошибка в грануляции объема. Необходимо прекратить все попытки торговых операций и изменить логику программы.</p>
ERR_MARKET_CLOSED	132	<p>Рынок закрыт. Можно повторить попытку через достаточно большой промежуток времени (от нескольких минут).</p>

ERR_TRADE_DISABLED	133	Торговля запрещена. Необходимо прекратить все попытки торговых операций.
ERR_NOT_ENOUGH_MONEY	134	Недостаточно денег для совершения операции. Повторять сделку с теми же параметрами нельзя. Попытку можно повторить после задержки от 5 секунд, уменьшив объем, но надо быть уверенным в достаточности средств для совершения операции.
ERR_PRICE_CHANGED	135	Цена изменилась. Можно без задержки обновить <u>данные</u> при помощи функции <u>RefreshRates</u> и повторить попытку.
ERR_OFF_QUOTES	136	Нет цен. Брокер по какой-то причине (например, в начале сессии цен нет, неподтвержденные цены, быстрый рынок) не дал цен или отказал. Необходимо после задержки от 5 секунд обновить <u>данные</u> при помощи функции <u>RefreshRates</u> и повторить попытку.
ERR_REQUOTE	138	Запрошенная цена устарела, либо перепутаны bid и ask. Можно без задержки обновить <u>данные</u> при помощи функции <u>RefreshRates</u> и повторить

		попытку. Если ошибка не исчезает, необходимо прекратить все попытки торговых операций и изменить логику программы.
ERR_ORDER_LOCKED	139	Ордер заблокирован и уже обрабатывается. Необходимо прекратить все попытки торговых операций и изменить логику программы.
ERR_LONG_POSITIONS_ONLY_ALLOWED	140	Разрешена только покупка. Повторять операцию SELL нельзя.
ERR_TOO_MANY_REQUESTS	141	Слишком много запросов. Необходимо уменьшить частоту запросов, изменить логику программы.
	142	Ордер поставлен в очередь. Это не ошибка, а один из кодов взаимодействия между клиентским терминалом и торговым сервером. Этот код может быть получен в редком случае, когда во время выполнения торговой операции произошёл обрыв и последующее восстановление связи. Необходимо обрабатывать так же как и ошибку 128.
	143	Ордер принят дилером к

		исполнению. Один из кодов взаимодействия между клиентским терминалом и торговым сервером. Может возникнуть по той же причине, что и код 142. Необходимо обрабатывать так же как и ошибку 128.
	144	Ордер аннулирован самим клиентом при ручном подтверждении сделки. Один из кодов взаимодействия между клиентским терминалом и торговым сервером.
ERR_TRADE_MODIFY_DENIED	145	Модификация запрещена, так как ордер слишком близок к рынку и заблокирован из-за возможного скорого исполнения. Можно не ранее, чем через 15 секунд, обновить <u>данные</u> при помощи функции <u>RefreshRates</u> и повторить попытку.
ERR_TRADE_CONTEXT_BUSY	146	Подсистема торговли занята. Повторить попытку только после того, как функция <u>IsTradeContextBusy</u> вернет FALSE.
ERR_TRADE_EXPIRATION_DENIED	147	Использование даты истечения ордера запрещено брокером. Операцию можно повторить

		только в том случае, если обнулить параметр expiration.
ERR_TRADE_TOO_MANY_ORDERS	148	Количество открытых и отложенных ордеров достигло предела, установленного брокером. Новые открытые позиции и отложенные ордера возможны только после закрытия или удаления существующих позиций или ордеров.
ERR_TRADE_HEDGE_PROHIBITED	149	Попытка открыть противоположную позицию к уже существующей в случае, если хеджирование запрещено. Сначала необходимо закрыть существующую противоположную позицию, либо отказаться от всех попыток таких торговых операций, либо изменить логику программы.
ERR_TRADE_PROHIBITED_BY_FIFO	150	Попытка закрыть позицию по инструменту в противоречии с правилом FIFO. Сначала необходимо закрыть более ранние существующие позиции по данному инструменту, либо отказаться от всех попыток таких торговых операций, либо изменить логику программы.

## 20.2 OrderClose

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

Закрытие позиции. Возвращает TRUE при успешном завершении функции. Возвращает FALSE при неудачном завершении функции. Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

### Параметры:

- ticket** - Уникальный порядковый номер ордера.
- lots** - Количество лотов для закрытия.
- price** - Цена закрытия.
- slippage** - Значение максимального проскальзывания в пунктах.
- Color** - Цвет стрелки закрытия на графике. Если параметр отсутствует или его значение равно CLR\_NONE, то стрелка на графике не отображается.

### Пример:

```
if (iRSI(NULL, 0, 14, PRICE_CLOSE, 0) > 75)  
{  
    OrderClose(order_id, 1, Ask, 3, Red);  
    return(0);  
}
```

## 20.3 OrderCloseBy

```
bool          int ticket, int opposite,  
OrderCloseBy( color Color=CLR_NONE)
```

Закрытие одной открытой позиции другой позицией, открытой по тому же самому инструменту, но в противоположном направлении. Возвращает TRUE при успешном завершении функции. Возвращает FALSE при неудачном завершении функции. Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

### Параметры:

- ticket** - Уникальный порядковый номер закрываемого ордера.
- opposite** - Уникальный порядковый номер противоположного ордера.
- Color** - Цвет стрелки закрытия на графике. Если параметр отсутствует или его значение равно CLR\_NONE, то стрелка на графике не

отображается.

**Пример:**

```
if (iRSI (NULL, 0, 14, PRICE_CLOSE, 0) > 75)
{
    OrderCloseBy (order_id, opposite_id);
    return (0);
}
```

## **20.4 OrderClosePrice**

**double OrderClosePrice ()**

Возвращает цену закрытия выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if (OrderSelect (10, SELECT_BY_POS, MODE_HISTORY) == true)
{
    datetime ctm = OrderOpenTime ();
    if (ctm > 0) Print ("Open time for the order 10 ", ctm);
    ctm = OrderCloseTime ();
    if (ctm > 0) Print ("Close time for the order 10 ", ctm);
}
else
    Print ("OrderSelect failed error code is", GetLastError ());
```

## **20.5 OrderCloseTime**

**datetime OrderCloseTime ()**

Возвращает время закрытия для выбранного ордера. Только закрытые ордера имеют время закрытия, не равное 0. Открытые или отложенные ордера имеют время закрытия, равное 0.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if (OrderSelect (10, SELECT_BY_POS, MODE_HISTORY) == true)
{
    datetime ctm = OrderOpenTime ();
    int ticket = OrderTicket ();
    if (ctm > 0) Print ("Время открытия ордера ? ", ticket, " ", ctm);
    ctm = OrderCloseTime ();
    if (ctm > 0) Print ("Время закрытия ордера ? ", ticket, " ", ctm);
}
else
    Print ("OrderSelect () вернул ошибку ", GetLastError ());
```

## 20.6 OrderComment

**string OrderComment()**

Возвращает комментарий для выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции [OrderSelect\(\)](#).

**Пример:**

```
string comment;
if(OrderSelect(10,SELECT_BY_TICKET)==false)
{
    Print("OrderSelect() вернул ошибку - ",GetLastError());
    return(0);
}
comment = OrderComment();
// ...
```

## 20.7 OrderCommission

**double OrderCommission()**

Возвращает значение рассчитанной комиссии для выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции [OrderSelect\(\)](#).

**Пример:**

```
if(OrderSelect(10,SELECT_BY_POS)==true)
    Print("Commission for the order 10 ",OrderCommission());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

## 20.8 OrderDelete

**bool OrderDelete(int ticket, color arrow\_color=CLR\_NONE)**

Удаляет ранее установленный отложенный ордер. Возвращает TRUE при успешном завершении функции. Возвращает FALSE при неудачном завершении функции. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

**Параметры:**

- ticket** - Уникальный порядковый номер ордера.
- arrow\_color** - Цвет стрелки на графике. Если параметр отсутствует или его значение равно CLR\_NONE, то стрелка на графике не отображаются.

**Пример:**

```
if(Ask>var1)
{
```

```
OrderDelete(order_ticket);
return(0);
}
```

## **20.9 OrderExpiration**

**datetime OrderExpiration()**

Возвращает дату истечения для выбранного отложенного ордера.

Ордер должен быть предварительно выбран с помощью функции [OrderSelect\(\)](#).

**Пример:**

```
if(OrderSelect(10, SELECT_BY_TICKET)==true)
    Print("Order expiration for the order #10 is ",OrderExpiration());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

## **20.10 OrderLots**

**double OrderLots()**

Возвращает количество лотов для выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции [OrderSelect\(\)](#).

**Пример:**

```
if(OrderSelect(10,SELECT_BY_POS)==true)
    Print("lots for the order 10 ",OrderLots());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

## **20.11 OrderMagicNumber**

**int OrderMagicNumber()**

Возвращает идентификационное ("магическое") число для выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции [OrderSelect\(\)](#).

**Пример:**

```
if(OrderSelect(10,SELECT_BY_POS)==true)
    Print("Magic number for the order 10 ", OrderMagicNumber());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

## **20.12 OrderModify**

**bool int ticket, double price, double stoploss,**

```
OrderModify( double takeprofit, datetime expiration,  
             color arrow_color=CLR_NONE)
```

Изменяет параметры ранее открытых позиций или отложенных ордеров. Возвращает TRUE при успешном завершении функции. Возвращает FALSE при неудачном завершении функции. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

Замечания: цену открытия и время истечения можно изменять только у отложенных ордеров.

Если в качестве параметров функции передать неизменные значения, то в этом случае будет сгенерирована [ошибка 1](#) (ERR\_NO\_RESULT).

На некоторых торговых серверах может быть установлен запрет на применение срока истечения отложенных ордеров. В этом случае при попытке задать ненулевое значение в параметре *expiration* будет сгенерирована [ошибка 147](#) (ERR\_TRADE\_EXPIRATION\_DENIED).

#### Параметры:

<b>ticket</b>	- Уникальный порядковый номер ордера.
<b>price</b>	- Новая цена открытия отложенного ордера.
<b>stoploss</b>	- Новое значение StopLoss.
<b>takeprofit</b>	- Новое значение TakeProfit.
<b>expiration</b>	- Время истечения отложенного ордера.
<b>arrow_color</b>	- Цвет стрелок модификации StopLoss и/или TakeProfit на графике. Если параметр отсутствует или его значение равно CLR_NONE, то стрелки на графике не отображаются.

#### Пример:

```
if(TrailingStop>0)  
{  
    OrderSelect(12345,SELECT_BY_TICKET);  
    if(Bid-OrderOpenPrice()>Point*TrailingStop)  
    {  
        if(OrderStopLoss()<Bid-Point*TrailingStop)  
        {  
            OrderModify(OrderTicket(),OrderOpenPrice(),Bid-  
Point*TrailingStop,OrderTakeProfit(),0,Blue);  
            return(0);  
        }  
    }  
}
```

### **20.13 OrderOpenPrice**

**double OrderOpenPrice()**

Возвращает цену открытия для выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if(OrderSelect(10, SELECT_BY_POS)==true)
    Print("open price for the order 10 ",OrderOpenPrice());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

### **20.14 OrderOpenTime**

**datetime OrderOpenTime()**

Возвращает время открытия выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if(OrderSelect(10, SELECT_BY_POS)==true)
    Print("open time for the order 10 ", OrderOpenTime());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

### **20.15 OrderPrint**

**void OrderPrint()**

Выводит данные ордера в журнал в виде строки следующего формата:

номер тикета; время открытия; торговая операция; количество лотов; цена открытия; стоп лосс; тейк профит; время закрытия; цена закрытия; комиссия; своп; прибыль; комментарий; магическое число; дата истечения отложенного ордера.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if(OrderSelect(10, SELECT_BY_TICKET)==true)
    OrderPrint();
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

## 20.16 OrderProfit

**double OrderProfit()**

Возвращает значение чистой прибыли (без учёта свопов и комиссий) для выбранного ордера. Для открытых позиций это - текущая нереализованная прибыль. Для закрытых ордеров - зафиксированная прибыль.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if(OrderSelect(10, SELECT_BY_POS)==true)
    Print("Profit for the order 10 ",OrderProfit());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

## 20.17 OrderSelect

**bool OrderSelect(int index, int select, int pool=MODE\_TRADES)**

Функция выбирает ордер для дальнейшей работы с ним. Возвращает TRUE при успешном завершении функции. Возвращает FALSE при неудачном завершении функции. Чтобы получить информацию об ошибке, необходимо вызвать функцию GetLastError().

Параметр *pool* игнорируется, если ордер выбирается по номеру тикета. Номер тикета является уникальным идентификатором ордера. Чтобы определить, из какого списка выбран ордер, необходимо проанализировать его время закрытия. Если время закрытия ордера равно 0, то ордер является открытым или отложенным и взят из списка открытых позиций терминала. Отличить открытую позицию от отложенного ордера можно по типу ордера. Если время закрытия ордера не равно 0, то ордер является закрытым или удаленным отложенным и был выбран из истории терминала. Отличить закрытый ордер от удаленного отложенного также можно по типу ордера.

**Параметры:**

- index** - Позиция ордера или номер ордера в зависимости от второго параметра.
- select** - Флаг способа выбора. Может быть одним из следующих величин:  
SELECT\_BY\_POS - в параметре *index* передается порядковый номер позиции в списке,  
SELECT\_BY\_TICKET - в параметре *index* передается номер тикета.
- pool** - Источник данных для выбора. Используется, когда параметр *select* равен SELECT\_BY\_POS. Может быть одной из следующих величин:

MODE\_TRADES (по умолчанию) - ордер выбирается среди открытых и отложенных ордеров,  
MODE\_HISTORY - ордер выбирается среди закрытых и удаленных ордеров.

**Пример:**

```
if(OrderSelect(12470, SELECT_BY_TICKET)==true)
{
    Print("order #12470 open price is ", OrderOpenPrice());
    Print("order #12470 close price is ", OrderClosePrice());
}
else
    Print("OrderSelect() вернул ошибку - ", GetLastError());
```

### **20.18 OrderSend**

```
int          string symbol, int cmd, double volume, double price,
OrderSend( int slippage, double stoploss, double takeprofit,
           string comment=NULL, int magic=0,
           datetime expiration=0, color arrow_color=CLR_NONE)
```

Основная функция, используемая для открытия позиции или установки отложенного ордера.

Возвращает номер тикета, который назначен ордеру торговым сервером или -1 в случае неудачи. Чтобы получить дополнительную информацию об ошибке, необходимо вызвать функцию GetLastError().

Замечания.

При открытии рыночного ордера (OP\_SELL или OP\_BUY) в качестве цены открытия могут использоваться только самые последние цены Bid (для продажи) или Ask (для покупки). Если операция проводится по финансовому инструменту, отличному от текущего, то для получения последних котировок по этому инструменту необходимо воспользоваться функцией MarketInfo() с параметром MODE\_BID или MODE\_ASK. Нельзя использовать расчетную либо ненормализованную цену. Если запрашиваемой цены открытия не было в ценовом потоке либо запрашиваемая цена не нормализована в соответствии с количеством знаков после десятичной точки, то будет сгенерирована ошибка 129 (ERR\_INVALID\_PRICE). Если запрашиваемая цена открытия сильно устарела, то независимо от значения параметра *slippage* будет сгенерирована ошибка 138 (ERR\_REQUOTE). Если же запрашиваемая цена устарела, но ещё присутствует в ценовом потоке, то позиция открывается по текущей цене и только в том случае, если текущая цена

попадает в диапазон  $price \pm slippage$ .

Цены StopLoss и TakeProfit не могут располагаться слишком близко к рынку.

Минимальное расстояние стопов в пунктах можно получить, используя функцию MarketInfo() с параметром MODE\_STOPLEVEL. В случае ошибочных, а также ненормализованных стопов генерируется ошибка 130 (ERR\_INVALID\_STOPS).

При установке отложенного ордера цена открытия не может быть слишком близкой к рынку. Минимальное расстояние отложенной цены от текущей рыночной цены в пунктах также можно получить, используя функцию MarketInfo() с параметром MODE\_STOPLEVEL. В случае неправильной цены открытия отложенного ордера будет сгенерирована ошибка 130 (ERR\_INVALID\_STOPS).

На некоторых торговых серверах может быть установлен запрет на применение срока истечения отложенных ордеров. В этом случае при попытке задать ненулевое значение в параметре *expiration* будет сгенерирована ошибка 147 (ERR\_TRADE\_EXPIRATION\_DENIED).

На некоторых торговых серверах может быть установлен лимит на общее количество открытых и отложенных ордеров. При превышении этого лимита новая позиция открыта не будет (отложенный ордер не будет установлен), и торговый сервер вернет ошибку 148 (ERR\_TRADE\_TOO\_MANY\_ORDERS).

#### Параметры:

<b>symbol</b>	-	Наименование финансового инструмента, с которым проводится торговая операция.
<b>cmd</b>	-	Торговая операция. Может быть любым из значений <u>торговых операций</u> .
<b>volume</b>	-	Количество лотов.
<b>price</b>	-	Цена открытия.
<b>slippage</b>	-	Максимально допустимое отклонение цены для рыночных ордеров (ордеров на покупку или продажу).
<b>stoploss</b>	-	Цена закрытия позиции при достижении уровня убыточности (0 в случае отсутствия уровня убыточности).

<b>takeprofit</b>	-	Цена закрытия позиции при достижении уровня прибыльности (0 в случае отсутствия уровня прибыльности).
<b>comment</b>	-	Текст комментария ордера. Последняя часть комментария может быть изменена торговым сервером.
<b>magic</b>	-	Магическое число ордера. Может использоваться как определяемый пользователем идентификатор.
<b>expiration</b>	-	Срок истечения отложенного ордера.
<b>arrow_color</b>	-	Цвет открывающей стрелки на графике. Если параметр отсутствует или его значение равно CLR_NONE, то открывающая стрелка не отображается на графике.

**Пример:**

```
int ticket;
if(iRSI(NULL,0,14,PRICE_CLOSE,0)<25)
{
    ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,Bid-25*Point,Ask+25*Point,"My
order #"+counter,16384,0,Green);
    if(ticket<0)
    {
        Print("OrderSend failed with error #",GetLastError());
        return(0);
    }
}
```

## **20.19 OrdersHistoryTotal**

**int OrdersHistoryTotal()**

Возвращает количество закрытых позиций и удаленных ордеров в истории текущего счета, загруженной в клиентском терминале. Размер списка истории зависит от текущих настроек вкладки "История счета" терминала.

**Пример:**

```
// retrieving info from trade history
int i,accTotal=OrdersHistoryTotal();
for(i=0;i<accTotal;i++)
{
    //---- check selection result
    if(OrderSelect(i,SELECT_BY_POS,MODE_HISTORY)==false)
    {
        Print("Ошибка при доступе к исторической базе
(",GetLastError(),")");
        break;
    }
}
```

```
// работа с ордером ...  
}
```

## 20.20 OrderStopLoss

**double OrderStopLoss()**

Возвращает значение цены закрытия позиции при достижении уровня убыточности (stop loss) для текущего выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if(OrderSelect(ticket,SELECT_BY_POS)==true)  
    Print("Stop loss value for the order 10 ", OrderStopLoss());  
else  
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

## 20.21 OrdersTotal

**int OrdersTotal()**

Возвращает общее количество открытых и отложенных ордеров.

**Пример:**

```
int handle=FileOpen("OrdersReport.csv",FILE_WRITE|FILE_CSV,"\t");  
if(handle<0) return(0);  
// запишем заголовок в файл  
FileWrite(handle,"#", "Цена открытия", "Время открытия", "Символ", "Лоты");  
int total=OrdersTotal();  
// записываем в файл только открытые ордера  
for(int pos=0;pos<total;pos++)  
{  
    if(OrderSelect(pos,SELECT_BY_POS,MODE_TRADES)==false) continue;  
  
FileWrite(handle,OrderTicket(),OrderOpenPrice(),OrderOpenTime(),OrderSymbol()  
(),OrderLots());  
}  
FileClose(handle);
```

## 20.22 OrderSwap

**double OrderSwap()**

Возвращает значение свопа для текущего выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if(OrderSelect(order_id, SELECT_BY_TICKET)==true)  
    Print("Swap for the order #", order_id, " ",OrderSwap());  
else
```

```
Print("OrderSelect() вернул ошибку - ",GetLastError());
```

### **20.23 OrderSymbol**

**string OrderSymbol()**

Возвращает наименование финансового инструмента для текущего выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if(OrderSelect(12, SELECT_BY_POS)==true)
    Print("symbol of order #", OrderTicket(), " is ", OrderSymbol());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

### **20.24 OrderTakeProfit**

**double OrderTakeProfit()**

Возвращает значение цены закрытия позиции при достижении уровня прибыльности (take profit) для текущего выбранного ордера

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if(OrderSelect(12, SELECT_BY_POS)==true)
    Print("Order #",OrderTicket()," profit: ", OrderTakeProfit());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

### **20.25 OrderTicket**

**int OrderTicket()**

Возвращает номер тикета для текущего выбранного ордера.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
if(OrderSelect(12, SELECT_BY_POS)==true)
    Print("Order #",OrderTicket()," profit: ", OrderTakeProfit());
else
    Print("OrderSelect() вернул ошибку - ",GetLastError());
```

### **20.26 OrderType**

**int OrderType()**

Возвращает тип операции текущего выбранного ордера. Может быть одной из следующих величин:

OP\_BUY - позиция на покупку,

OP\_SELL - позиция на продажу,

OP\_BUYLIMIT - отложенный ордер на покупку по достижении заданного уровня, текущая цена выше уровня,

OP\_BUYSTOP - отложенный ордер на покупку по достижении заданного уровня, текущая цена ниже уровня,

OP\_SELLLIMIT - отложенный ордер на продажу по достижении заданного уровня, текущая цена ниже уровня,

OP\_SELLSTOP - отложенный ордер на продажу по достижении заданного уровня, текущая цена выше уровня.

Ордер должен быть предварительно выбран с помощью функции OrderSelect().

**Пример:**

```
int order_type;
if(OrderSelect(12, SELECT_BY_POS)==true)
{
    order_type=OrderType();
    // ...
}
else
    Print("OrderSelect() вернул ошибку - ", GetLastError());
```

## 21 Операции с графиками

Группа функций, предназначенных для работы с окном текущего графика.

### 21.1 HideTestIndicators

```
void HideTestIndicators (bool hide)
```

Функция выставляет флаг скрытия индикаторов, вызываемых экспертом. При открытии графика после тестирования индикаторы, помеченные флагом скрытия, не будут выведены на график тестирования. Перед каждым вызовом индикатор помечается текущим установленным флагом скрытия.

Необходимо отметить, что на график тестирования могут быть выведены только те индикаторы, которые непосредственно вызываются из тестируемого эксперта.

**Параметры:**

**hide** - TRUE - если нужно прятать индикаторы, иначе FALSE.

**Пример:**

```
HideTestIndicators (true);  
MaCurrent=iMA (NULL, 0, 56, 0, MODE_EMA, PRICE_CLOSE, 0);  
MaPrevious=iMA (NULL, 0, 56, 0, MODE_EMA, PRICE_CLOSE, 1);  
HideTestIndicators (false);
```

### 21.2 Period

```
int Period()
```

Возвращает значение числа минут периода для текущего графика.

**Пример:**

```
Print ("Период ", Period());
```

### 21.3 RefreshRates

```
bool RefreshRates ()
```

Обновление данных в предопределенных переменных и массивах-таймсериях. Эта функция используется, когда эксперт или скрипт производит вычисления в течение долгого времени и нуждается в обновленных данных. Возвращается TRUE, если данные обновлены, иначе FALSE. Данные могут не обновиться только по той причине, что они соответствуют текущему состоянию клиентского терминала. Эксперты и скрипты работают с собственной копией исторических данных. Копия данных по текущему инструменту создается при первоначальном запуске эксперта или скрипта. При каждом следующем запуске эксперта (напомним, что скрипт выполняется однократно и не зависит

от приходящих тиков) первоначально созданная копия обновляется. За то время, пока эксперт или скрипт работает, может прийти один или несколько новых тиков, поэтому данные могут устареть.

**Пример:**

```
int ticket;
while(true)
{
    ticket=OrderSend(Symbol(),OP_BUY,1.0,Ask,3,0,0,"комментарий
эксперта",255,0,CLR_NONE);
    if(ticket<=0)
    {
        int error=GetLastError();
        //---- недостаточно денег
        if(error==134) break;
        //---- 10 секунд ожидания
        Sleep(10000);
        //---- обновим ценовые данные
        RefreshRates();
    }
    else
    {
        OrderSelect(ticket,SELECT_BY_TICKET);
        OrderPrint();
        break;
    }
}
```

## **21.4 Symbol**

**string Symbol()**

Возвращает текстовую строку с именем текущего финансового инструмента.

**Пример:**

```
int total=OrdersTotal();
for(int pos=0;pos<total;pos++)
{
    // результат выбора проверки, так как ордер может быть закрыт или
    удален в это время!
    if(OrderSelect(pos, SELECT_BY_POS)==false) continue;
    if(OrderType()>OP_SELL || OrderSymbol()!=Symbol()) continue;
    // делает некоторую обработку...
}
```

## **21.5 WindowBarsPerChart**

**int WindowBarsPerChart()**

Функция возвращает количество баров, помещающихся в окно текущего графика.

**Пример:**

```

// обработка видимых баров.
int bars_count=WindowBarsPerChart();
int bar=WindowFirstVisibleBar();
for(int i=0; i<bars_count; i++,bar--)
{
    // номера баров уменьшаются, так как нумерация идет в обратном
    // порядке.
    // ...
}

```

## 21.6 WindowExpertName

**string WindowExpertName()**

Возвращает имя выполняющегося эксперта, скрипта, пользовательского индикатора или библиотеки, в зависимости от того, из какой MQL4-программы вызвана данная функция.

**Пример:**

```

string name=WindowExpertName();
GlobalVariablesDeleteAll(name);

```

## 21.7 WindowFind

**int WindowFind(string name)**

Возвращает номер подокна графика, содержащего индикатор с указанным именем *name*, если он найден, иначе возвращается -1.

WindowFind() возвращает -1, если пользовательский индикатор ищет сам себя в процессе инициализации *init()*.

**Параметры:**

**name** - Короткое имя индикатора.

**Пример:**

```

int win_idx=WindowFind("MACD(12,26,9)");

```

## 21.8 WindowFirstVisibleBar

**int WindowFirstVisibleBar()**

Функция возвращает номер первого видимого бара в окне текущего графика. Необходимо иметь в виду, что ценовые бары нумеруются задом наперед, от последнего к первому. Текущий бар, самый последний в ценовом массиве, имеет индекс 0. Самый старый бар имеет индекс Bars-1. Если номер первого видимого бара меньше, чем количество видимых баров на графике на 2 и более, это значит, что окно графика заполнено не до конца и имеется поле справа.

**Пример:**

```

// обработка видимых баров.
int bars_count=WindowBarsPerChart();
int bar=WindowFirstVisibleBar();
for(int i=0; i<bars_count; i++,bar--)
{
    // номера баров уменьшаются, так как нумерация идет в обратном
    // порядке.
    // ...
}

```

## 21.9 WindowHandle

**int WindowHandle(string symbol, int timeframe)**

Возвращает системный дескриптор окна (window handle), содержащего указанный график. Если график с *symbol* и *timeframe* на момент вызова функции не открыт, то возвращается 0.

**Параметры:**

**symbol** - Символ.  
**timeframe** - Период. Может быть одним из периодов графика.

**Пример:**

```
int win_handle=WindowHandle("USDX",PERIOD_H1);
if(win_handle!=0)
    Print("Окно с USDX,H1 обнаружено. Массив будет скопирован.");
```

## 21.10 WindowVisible

**bool WindowVisible(int index)**

Возвращает TRUE, если подокно графика видимо, иначе возвращает FALSE. Подокно графика может быть скрыто из-за свойств видимости помещенного в него индикатора.

**Параметры:**

**index** - Индекс подокна графика.

**Пример:**

```
int maywin=WindowFind("MyMACD");
if(maywin>-1 && WindowVisible(maywin)==true)
    Print("окно MyMACD видимое");
else
    Print("окно MyMACD не обнаружено или не видимое");
```

## 21.11 WindowOnDropped

**int WindowOnDropped()**

Возвращает индекс окна, в которое был брошен эксперт, пользовательский индикатор или скрипт. Это значение будет верным только в том случае, если эксперты, пользовательские индикаторы и скрипты прикреплены с помощью мыши (технология "drag and drop").

Для пользовательских индикаторов, находящихся в процессе инициализации (вызов из функции *init()*) этот индекс не определен.

Возвращаемый индекс является номером окна (0-главное окно графика, подокна индикаторов нумеруются с 1), в котором работает пользовательский индикатор. В процессе инициализации пользовательский индикатор может создать свое собственное новое подокно и его номер будет отличаться от номера окна, на которое действительно был брошен индикатор.

См. также [WindowXOnDropped\(\)](#), [WindowYOnDropped\(\)](#)

**Пример:**

```
if(WindowOnDropped()!=0)
{
    Print("Скрипт MyVisualTrading должен быть брошен на основное окно графика!");
    return(0);
}
```

## 21.13 WindowPriceMax

**double WindowPriceMax(int index=0)**

Возвращает максимальное значение вертикальной шкалы указанного подокна текущего графика (0-главное окно графика, подокна индикаторов нумеруются с 1). Если индекс подокна не указан, то возвращается максимальное значение ценовой шкалы главного окна графика.

См. также [WindowPriceMin\(\)](#), [WindowFirstVisibleBar\(\)](#), [WindowBarsPerChart\(\)](#)

**Параметры:**

**index** - Индекс подокна текущего графика (0 - основной график цены).

**Пример:**

```

double top=WindowPriceMax();
double bottom=WindowPriceMin();
datetime left=Time[WindowFirstVisibleBar()];
int right_bound=WindowFirstVisibleBar()-WindowBarsPerChart();
if(right_bound<0) right_bound=0;
datetime right=Time[right_bound]+Period()*60;
//----
ObjectCreate("Padding_rect",OBJ_RECTANGLE,0,left,top,right,bottom);
ObjectSet("Padding_rect",OBJPROP_BACK,true);
ObjectSet("Padding_rect",OBJPROP_COLOR,Blue);
WindowRedraw();

```

## 21.14 WindowPriceMin

**double WindowPriceMin(int index=0)**

Возвращает минимальное значение вертикальной шкалы указанного подокна текущего графика (0-главное окно графика, подокна индикаторов нумеруются с 1). Если индекс подокна не указан, то возвращается минимальное значение ценовой шкалы главного окна графика.

См. также [WindowPriceMax\(\)](#), [WindowFirstVisibleBar\(\)](#), [WindowBarsPerChart\(\)](#)

**Параметры:**

**index** - Индекс подокна текущего графика (0 - основной график цены).

**Пример:**

```

double top=WindowPriceMax();
double bottom=WindowPriceMin();
datetime left=Time[WindowFirstVisibleBar()];
int right_bound=WindowFirstVisibleBar()-WindowBarsPerChart();
if(right_bound<0) right_bound=0;
datetime right=Time[right_bound]+Period()*60;
//----
ObjectCreate("Padding_rect",OBJ_RECTANGLE,0,left,top,right,bottom);
ObjectSet("Padding_rect",OBJPROP_BACK,true);
ObjectSet("Padding_rect",OBJPROP_COLOR,Blue);
WindowRedraw();

```

## 21.15 WindowPriceOnDropped

**double WindowPriceOnDropped()**

Возвращает значение цены в точке графика, на которой был брошен эксперт или скрипт. Значение будет верным только в случае, если эксперт или скрипт перемещены с помощью мыши (технология "drag and drop").

Для пользовательских индикаторов это значение не определено.

См. также [WindowTimeOnDropped\(\)](#), [WindowYOnDropped\(\)](#), [WindowOnDropped\(\)](#)

**Пример:**

```

double drop_price=WindowPriceOnDropped();
datetime drop_time=WindowTimeOnDropped();
//---- может быть неопределенным (нуль)
if(drop_time>0)
{
    ObjectCreate("Уровень цены", OBJ_HLINE, 0, drop_price);
    ObjectCreate("Значение времени", OBJ_VLINE, 0, drop_time);
}

```

## 21.16 WindowRedraw

**void WindowRedraw()**

Принудительно перерисовывает текущий график. Обычно применяется после изменения свойств объектов.

**Пример:**

```
//---- установим новые свойства нескольких объектов
ObjectMove(object_name1, 0, Time[index], price);
ObjectSet(object_name1, OBJPROP_ANGLE, angle*2);
ObjectSet(object_name1, OBJPROP_FONTSIZE, fontsize);
ObjectSet(line_name, OBJPROP_TIME2, time2);
ObjectSet(line_name, OBJPROP_ANGLE, line_angle);
//---- теперь все перерисуем
WindowRedraw();
```

## 21.17 WindowScreenShot

**bool WindowScreenShot( string filename, int size\_x, int size\_y, int start\_bar=-1, int chart\_scale=-1, int chart\_mode=-1)**

Сохраняет изображение текущего графика в файле формата GIF. В случае неудачи возвращает FALSE. Чтобы получить информацию об ошибке, необходимо вызвать функцию [GetLastError\(\)](#).

Скриншот сохраняется в папке *каталог\_терминала\experts\files* (*каталог\_терминала\tester\files* в случае тестирования эксперта) или ее подпапках.

**Параметры:**

- filename** - Имя файла для скриншота.
- size\_x** - Ширина скриншота в пикселах.
- size\_y** - Высота скриншота в пикселах.
- start\_bar** - Номер первого видимого бара на скриншоте. Если указано значение 0, то скриншот снимается с текущего первого видимого бара. Если значение не указано, или указано отрицательное значение, то делается скриншот конца графика с учётом правого отступа.
- chart\_scale** - Масштаб графика, выводимого на скриншот. Может принимать значение от 0 до 5. Если значение не указано, или указано отрицательное значение, то используется текущий масштаб графика.
- chart\_mode** - Вид отображения графика. Может принимать значения: CHART\_BAR (0 - последовательность баров), CHART\_CANDLE (1 - японские свечи), CHART\_LINE (2 - линия по ценам закрытия). Если значение не указано, или указано отрицательное значение, то график выводится в своем текущем виде.

**Пример:**

```
int lasterror=0;
//---- тестер закрыл одну или несколько позиций
if(IsTesting() && ExtTradesCounter<TradesTotal())
{
    //---- снимем скриншот для проверки
    if(!WindowScreenShot("shots\\tester"+ExtShotsCounter+".gif",640,480))
        lasterror=GetLastError();
    else ExtShotsCounter++;
    ExtTradesCounter=TradesTotal();
}
```

## 21.18 WindowTimeOnDropped

**datetime WindowTimeOnDropped()**

Возвращает значение времени в точке графика, на которой был брошен эксперт или скрипт. Значение будет верным только в случае, если эксперт или скрипт перемещены с помощью мыши (технология "drag and drop").

Для пользовательских индикаторов это значение не определено.

См. также [WindowPriceOnDropped\(\)](#), [WindowXOnDropped\(\)](#), [WindowOnDropped\(\)](#)

**Пример:**

```
double drop_price=WindowPriceOnDropped();
datetime drop_time=WindowTimeOnDropped();
//---- может быть неопределенным (нуль)
if(drop_time>0)
{
    ObjectCreate("Уровень цены", OBJ_HLINE, 0, drop_price);
    ObjectCreate("Значение времени", OBJ_VLINE, 0, drop_time);
}
```

## **21.19 WindowsTotal**

**int WindowsTotal()**

Возвращает количество окон индикаторов на графике, включая главное окно графика.

**Пример:**

```
Print("Количество окон = ", WindowsTotal());
```

## **21.20 WindowXOnDropped**

**int WindowXOnDropped()**

Возвращает значение координаты по оси X в пикселах точки клиентской области окна графика, на которой был брошен эксперт или скрипт. Значение будет верным только в случае, если эксперт или скрипт перемещены с помощью мыши (технология "drag and drop").

См. также [WindowYOnDropped\(\)](#), [WindowTimeOnDropped\(\)](#), [WindowOnDropped\(\)](#)

**Пример:**

```
Print("Эксперт был прикреплен к окну в точке с координатами
x=",WindowXOnDropped()," y=",WindowYOnDropped());
```

## **21.21 WindowYOnDropped**

**int WindowYOnDropped()**

Возвращает значение координаты по оси Y в пикселах точки клиентской области окна графика, на которой был брошен эксперт или скрипт. Значение будет верным только в случае, если эксперт или скрипт перемещены с помощью мыши (технология "drag and drop").

См. также [WindowXOnDropped\(\)](#), [WindowPriceOnDropped\(\)](#), [WindowOnDropped\(\)](#)

**Пример:**

```
Print("Эксперт был прикреплен к окну в точке с координатами
x=",WindowXOnDropped()," y=",WindowYOnDropped());
```

## 22 Устаревшие функции

В процессе дальнейшей разработки языка MQL4 в целях систематизации некоторые функции были переименованы и перемещены из одной группы в другую. Старые названия функций не подсвечиваются и не связаны со справкой MetaEditor. Старые названия функций можно использовать, так как компилятор правильно их воспримет. Однако мы настоятельно рекомендуем использовать новые названия.

Старое название	Новое название
BarsPerWindow	<u>WindowBarsPerChart</u>
ClientTerminalName	<u>TerminalName</u>
CurTime	<u>TimeCurrent</u>
CompanyName	<u>TerminalCompany</u>
FirstVisibleBar	<u>WindowFirstVisibleBar</u>
Highest	<u>iHighest</u>
HistoryTotal	<u>OrdersHistoryTotal</u>
LocalTime	<u>TimeLocal</u>
Lowest	<u>iLowest</u>
ObjectsRedraw	<u>WindowRedraw,</u>
PriceOnDropped	<u>WindowPriceOnDropped</u>
ScreenShot	<u>WindowScreenShot</u>
ServerAddress	<u>AccountServer</u>
TimeOnDropped	<u>WindowTimeOnDropped</u>